# The user perceived performance of route planning APIs

Bert Marcelis

Supervisor(s): Prof. dr. ir. Ruben Verborgh, dr. ing. Pieter Colpaert, Julian Andres Rojas Melendez

*Abstract*—For the Web architectures behind route planning applications, remote procedure calls are commonplace, in which a server calculates the routes for all end-users. Linked Connections introduces an alternative architecture following the REST constraints, publishing the raw data in fragments. While benchmarks show a higher cost-efficiency of Linked Connections on the server, it is currently not known how it performs on clients, that now also need to execute the route planning algorithm. In this work, we study the user-perceived performance of route planning on the client-side, consuming more bandwidth, versus route planning on the server-side. An isomorphic app was developed with both route planning on the client-side as on the server-side. Both technical performance and user-perceived performance were tested among 17 travelers, and 81 respondents gave insights on their use of travel companions. We found that the performance of Linked Connections heavily depends on the type of information, the query and the user's device. Linked Connections can be faster or slower than a query answering API based on these parameters. For a majority of the users, the benefit of offline searches outweighs the slower speed of Linked Connections and even though Linked Connections is slower than the reference API, users consider it as fast as their default application on recent devices.

*Keywords*— Linked Connections, public transport

## I. Introduction

**P**UBLIC Public transport is an essential aspect of modern cities. To make use of it, travellers frequently use their smartphones (applications) or public transport companies' websites to get route planning advice. Today most of these mobile and Web applications follow a client-server architecture that makes use of Remote Procedure Call (RPC) APIs for exchanging data. In order to handle high volumes of requests, such an approach requires high levels of investment on computational infrastructure and also depends on a continuous connection between the client and the route planning server to answer any given query. A different approach consists on providing clients with all the necessary data of a given public transport network for them to be able to answer any query by themselves. This can be done by using the General Transport Format Specification (GTFS) and can be particularly useful when queries in bulk are needed, for example in insights building and analytics collecting applications. However this approach requires too much time and processing power to be applied in mobile end user devices.

Linked Connections emerges as an alternative approach that follows the REST constraints and consists on publishing the raw data in fragments, sorted in a timely fashion. Clients can download these data fragments and perform route-planning algorithms on top of them. Previous research has proven Linked Connections to be 75% more cost-efficient [1] than RPC-based approaches. However, while performance and costs on server-side have been researched,

the performance and user-experience for a client implementing Linked Connections are still unknown. In this work we explore various aspects of the performance and user-experience of a client-side Linked Connections-based application through the development of an isomorphic mobile application that compares the Linked Connections approach to a reference RPC API, which uses the same data and algorithms, but runs all calculations on server-side.

The remainder of this article is structured as follows. We first describe the state of the art, we then describe the meaning of user-perceived performance on Web architectures. Afterwards we describe the evaluation setup and the design choices. Then we present the results obtained and finally we present the discussion and conclusions drawn from this work.

## II. State of the art

Application for public transport are, at this time, commonly driven by RPC APIs. These APIs are based on dumb clients asking a smart server for an answer. On one hand this means it can be used on all kinds of clients, as it does not require much resources client-side to offer good performance [?]. On the other hand it can be costly, as every response is personalized, the server has to do processor-intensive tasks and needs to be scaled according to the load. Other negative aspects, specific for route planning, are the requirements for a constant internet connection, and the fact that individual developers can't alter the route planning algorithm for their application. An RPC API can transfer data in various formats, for example using Simple Object Access Protocol (SOAP), JSON RPC or Representational State Transfer (REST).

When transport data is needed in bulk, for example when someone wants to gather statistics, GTFS can be used. GTFS is divided into two categories, General Transit Format Specifications Static (GTFS) and General Transit Format Specifications Realtime (GTFS-RT). A GTFS feed is composed of a series of text files collected in a ZIP file. Each file models a particular aspect of transit information: stops, routes, trips, and other schedule data [10]. Theses files link together in a similar fashion to a relational database. GTFS Realtime is a feed specification that allows public transportation agencies to provide realtime updates about their fleet to application developers. It is an extension to GTFS [11], and is not of much use on its own. While it is a resource-intensive task to deduce information from GTFS, it does not need API calls for every query. Unfortunately, due to GTFS being resource intensive, it's

not fit for mobile devices or cases where results are needed quickly.

Linked connections tries to find a balance between these two technologies, by publishing the raw data in chronological, easy-to-use linked fragments. It's proven to be server efficient, and when load increases, performance increases instead of decreases [1].

Both Linked Connections and GTFS require the implementation of a route planning algorithm in order to determine a route from A to B when needed. Route planning algorithms have been studied extensively in the past 50 years. Many algorithms solve this problem by using a graph, for example the Dijkstra algorithm [6] and the algorithm of Bellman-Ford [7]. These algorithms rely on a graph, modelled using neighbour-matrices or neighbour-lists. The Connection Scan Algorithm (CSA) does not rely on these matrices or lists, but requires a chronological list of all vehicle departures [8]. This makes CSA perfectly fit for Linked Connections.

### III. User perceived performance of web architectures

Every Web architecture has specific properties (e.g. latency, performance, cache reuse, etc) [4]. While these are all technical values, there is also the user-perceived performance, which is a subjective user experience. First defined by Roy T. Fielding [2] as the latency between steady-states for a browser-based application, it comprises all processing needed, not only network traffic related. For a browser application, the total latency consists of the following parts:
1. The time needed for the user agent to recognize the event that initiated the action
2. The time required to set-up any interaction(s) between components
3. The time required to transmit each interaction to the components
4. The time required to process each interaction on those components
5. The time required to complete sufficient transfer before the user agent is able to begin rendering a usable result.
6. The time required to complete processing of the result of the interaction(s) before the user agent is able to begin rendering a usable result.

While only steps 3, 4 and 5 are directly dependent on network connectivity, all of these steps can be influenced by the used architecture. When not only focussing on performance, but also on the way users interact with technology, it is important to define the user-experience (UX). UX is a broad term, originally designating the design and usage of interfaces, making it a synonym for interactions and usability. But it is also used for non-instrumental needs and experiences in a more complex way [3]. In this work UX will always be considered in this broader way.

### IV. Evaluation design

In order to evaluate Linked Connections and compare it to RPC APIs, an isomorphic application was developed. This application allows to test Linked Connections and an RPC API, while keeping the user interface and client-side logic identical. The RPC API, LC2Irail, is backed by the same data and algorithms as the client-side Linked Connections implementation, thus allowing a fair comparison. As a result, the only difference is which entity does the processing, and how data is exchanged.

Through this application, user testing was conducted on 17 test persons. Each tester was asked to search for data he/she usually searches for, one time using the reference RPC API as data provider, and using the Linked Connections implementation the second time. After testing and grading Linked Connections, they were also encouraged to turn off their network connection and test the off-line capabilities using Linked Connections. Users were only informed about the differences between both implementations after the test was completed, to prevent any type of bias on them. For each implementation, the user graded the listing speed of departures and arrivals for a station (liveboards), journeys from A to B (routes) and vehicle trips (vehicles). Users were also asked to compare each implementation to the application they usually use. To conclude, they had to explicitly choose between both implementations regarding speed for each endpoint, and which implementation they favoured the most, taking into account both speed and offline access.

In order to assess which aspects were more important to users, a survey was held for 81 travellers. This survey asked about past experiences, needs, mobile data, privacy, and concluded by shortly polling how interested respondents were in the features offered by Linked Connections.

Along with subjective experiences from users, objective benchmarks were also executed. These benchmarks used a fraction of the real-world searches handled by the iRail API [9], to determine the speed needed to load a certain number of results. This was done both on a relatively modern smartphone (HTC 10, Android 8.0) and on an older smartphone (HTC One, Android 5.0). By running each benchmark on both devices, it was possible to evaluate the impact on the hardware of the Linked Connections approach.

For the local implementation, two versions were implemented. The first implementation relies on the default Android JSON parser, org.json. This implementation proved to be rather slow, with vehicle requests taking around four seconds on the HTC 10. It used an index containing the first departure of every vehicle and it did not use a cache. CPU profiling for this implementation revealed heavy CPU usage by the JSON parsing process, for which several optimizations were applied. These optimizations proved to be insufficient, which is why a second implementation was built. This new implementation makes use of the LoganSquare parser, resulting in a reduction of around one second for the same queries.

### V. Results

When testing both parsers with a cache memory on the device's flash storage, performance increased even more, going from around four seconds to just below two seconds. Based on this, the user-test was altered. While the first ten

users tested the first implementation, the last seven testers received the second implementation. The second group reported slightly faster experiences for routes and vehicles, but the separate test groups are too small to generalize this to the entire population. Users who tested both implementations reported an perceived speed gain, independent of the test device hardware. All benchmarks use the second (more efficient) implementation.
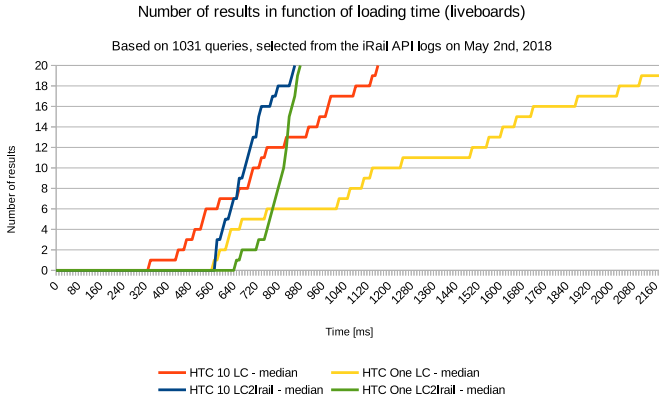


Fig. 1.

The incremental results for liveboards shown in figure 1, make clear that there are significant differences between both architectures. Linked Connections is faster than LC2Irail (the reference RPC API) on both devices for the first few results. However, when ten results are needed – about the number of results which fit on a large screen – LC2Irail is faster on both devices. It is clear that the RPC API performs similarly on both devices, but Linked Connections does not. There is a gap between the time needed by Linked Connections on both devices, which grows with the number of results needed.
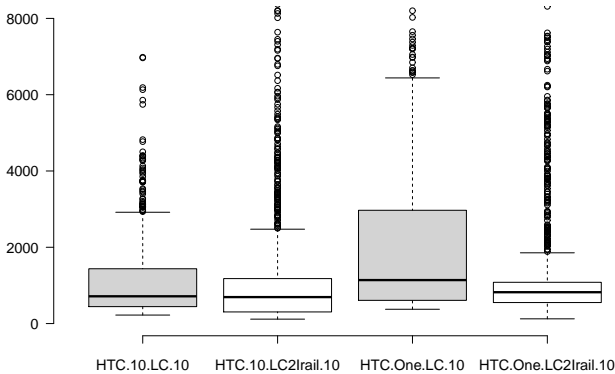


Fig. 2.

Taking a look at the distributions of the loading times for the tenth results (figure 2) it is confirmed again that LC2Irail performs the same on both devices. This contrasts with Linked Connections, which has a significantly

larger distribution of loading times on the HTC One. While the differences between the median response time of Linked Connections on the HTC 10 and LC2Irail on both devices cannot be distinguished by end users, there is a clear difference with the median of Linked Connections on the HTC One.
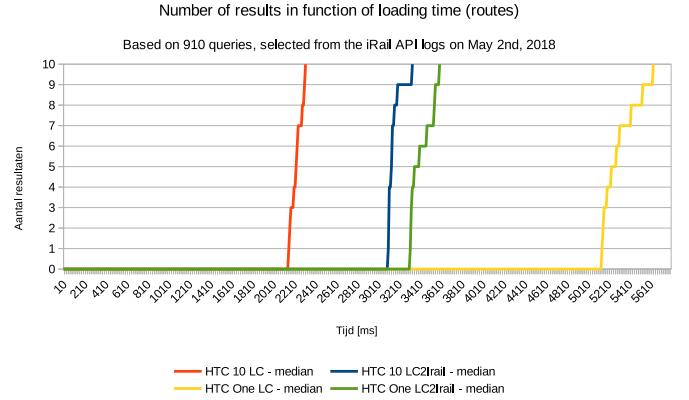


Fig. 3.

When looking at the incremental results for routes (figure 3) this difference between devices becomes even more visible. Linked Connections loads over 2 times faster on the HTC 10 compared to the HTC One. Due to this the fastest technique depends on the device used. Linked Connections performs better than LC2Irail on the modern HTC 10, but Linked Connections performs significantly worse than LC2Irail on the HTC One. LC2Irail performs about the same on both devices, which it also did for liveboards. Calculating routes relies heavily on the Connection Scan Algorithm, which makes it even more surprising to see that Linked Connections on the HTC 10 is faster than the RPC approach. This is also a possible explanation for the strong discrepancy between loading times of Linked Connections on both devices, as the slower processor and memory in the HTC One could hinder both the processing of the raw data, and the calculations using the Connection Scan Algorithm. Distributions for routes are shown in figure 4, where it becomes even more clear that LC2Irail sits inbetween the performance of Linked Connections on both devices.

The performance of LC2Irail for Vehicles is special compared to the previous two. Every vehicle trip is considered to be one atomic result, therefore incremental results are not supported for this data type. When looking at the distribution of the loading times (figure 5) it becomes clear that vehicles take a long time to load compared to other data structures, even though they are not obtained through complex algorithms. This data type needs a lot of data, however. LC2Irail, which has quick access to the data, performs a lot better here. Again, it performs consistently between devices, whereas Linked Connections needs two times as much time on the HTC One, compared to Linked Connections on the HTC 10.

Not only the data type and device affect the performance, but the exact query is of importance too. Calm
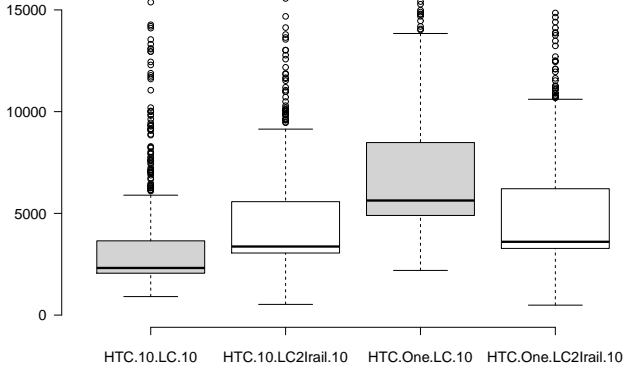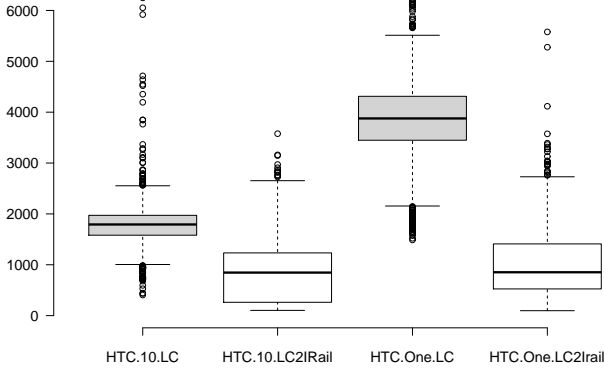
Fig. 4.



Fig. 5.



Fig. 6.

stations, long routes, or vehicles with a long trip take longer to load compared to busy stations, short routes or vehicles with a short trip. The time to load a number of results is directly related to the timespan in which the results can be found. When a larger timespan needs to be evaluated, the results will take longer to load.

The measured results seem to match the user experienced performance. This can be seen in figure 6. While quite some users still find Linked Connections faster for departures, less people choose Linked Connections for routes, and even less choose Linked Connections for vehicles. However, when users are asked to not only judge by the performance, but to also take additional features like offline access into account, a majority of the users who previously chose for LC2Irail changes their mind and chooses Linked Connections. Features like incremental results don't seem to improve the user-perceived performance – users keep waiting for all results to load before they start processing the results.
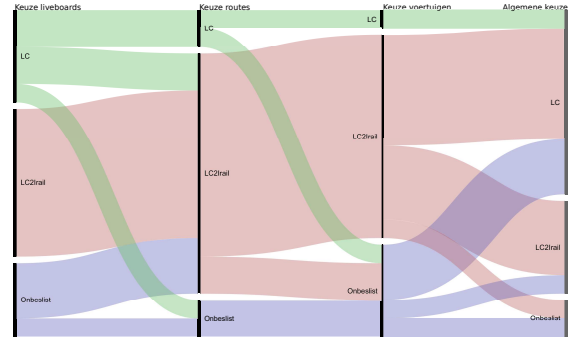
## VI. Discussion

LC2Irail is an ideal reference API. The performance is consistent across devices, and users are overwhelmingly positive over the performance. Consistent good performance, with a small spread of the user-perceived performance, is the goal of every application. Linked Connections seems unable to offer this. The user perceived performance depends heavily on the used device, the type of data which was requested, and on the exact query. Both measured performance and user-perceived performance are spread out, ranging from slower than LC2Irail to faster than LC2Irail, and from perceived quite slow to perceived extremely fast. It seems there is definitely potential in this technique, but heavy resource requirements are holding it back on older devices.

The difference between both Linked Connections implementations is an extremely important indicator. This difference shows that the performance of Linked Connections does not only depend on the device, data type and query, but also on the details of the implementation. When the parsing of both implementations is traced, the implementation based on LoganSquare is about two times slower than the original implementation, requiring 200 milliseconds to parse a single Linked Connections page, compared to 100 milliseconds when using the org.json parser. This is opposite to both benchmarks and user-perceived performance. The key to this mystery is found in the Garbage Collection (GC), ran by the Android Java Virtual Machine (JVM). The first implementation which uses the org.json parser creates more String objects than the second implementation based on LoganSquare. As String objects are immutable, creating too much of them or modifying them too much will trigger garbage collection, in which case the entire application is frozen as unreferenced objects are removed from the memory. Even further, the JVM garbage collection threshold depends on the heap size, which is configured by the device manufacturer and correlates with the available RAM and screen properties. This means older or cheaper devices are penalized twice: slower hardware means Linked Connections algorithms and GC take more time, while GC will be needed more often as the device has less memory. As a result, LoganSquare can be up to 8 times faster than org.jsonl for large payloads.

Knowing now the importance of an efficient implementation on all levels, it is possible to reimplement the Linked Connections client while removing or reducing the usage of Strings. This might lead to further performance improvements, potentially bringing the performance on older or cheaper devices up to par with performance on more powerful devices. Still, this forms a disadvantage for Linked Connections. Developers looking to implement this might not have the necessary (background) knowledge to optimize their implementation this thorough, or they might not have the time. This could form a barrier for adoption.

Another important curiosity is the U-turn most people made when asked to choose their favorite implementation, based on everything, including speed and offline access. While more and more people lean towards LC2Irail depending on the endpoint, as can be seen in figure 6, offline access seems to convince over half of the people picking LC2Irail to switch to Linked Connections. This can be explained by the general discontent regarding mobile network coverage while travelling by train, and people not having access to mobile data, or being afraid of using too much.

Parameters like data usage depend heavily on the implementation. Various optimizations are possible to reduce this, with offline access reducing this to zero.

A last important note is the performance after mass adoption. While the RPC API will suffer from reduced performance under high load, Linked Connections provides better performance as load increases. This means the Linked Connections results will only improve, while the performance of the RPC API can only decline.

## VII. Conclusion

While the reference RPC API provides excellent results, Linked Connections offers a wide range of performances depending on a lot of factors, sometimes performing better than this reference API, but often performing the same or worse. Users consistently perceive Linked Connections as slower, with only some outliers experiencing it faster. Performance heavily depends on the search, implementation and the users' device. This is a bad trait for an application, as the goal is to present a consistent experience across all devices.

There is, however, a high chance that Linked Connections can be improved further by paying attention to implementation details. A second performance gain could be obtained by the use of various indices, reducing the overhead of retrieving pages without useful data. This could also be achieved by running the algorithms on cached data first, after which the used pages could be retrieved from the server in order to obtain real-time information.

The performance of the route planning algorithm can be further optimized by first applying the Earliest Arrival Time algorithm, which will cache the required pages and determine the starting point where the Connection Scan Algorithm should start iterating over the pages. Search performance could also be improved by pre-loading data when it is highly likely the user will make a search. In this case it is no longer needed to retrieve the pages the moment the user confirms the search.

Another improvement could be obtained by first running the algorithm on cached or pre-loaded data. These results could be shown while real-time data is loaded from the server, after which the real-time data could be appended to the first results.

The last performance improvement could be achieved by keeping recently used pages in memory, thus reducing the amount of garbage, as it is likely that the next search will require data from the same page as the previous page. In reality, this might not be feasible due to constraints on the available memory. All performance improvements should take the device properties into account, as a device with limited RAM might be better off using a cache in the flash memory.

## References

[1] Pieter Colpaert, Publishing transport data for maximum reuse, Ph.D. dissertation, Ghent University, 2017. [Online]. Available: https://phd.pietercolpaert.be

[2] R. T. Fielding and R. N. Taylor, Principled design of the modern web architecture, 1999. [Online]. Available: https://www.ics.uci.edu/ fielding/pubs/fse99_webarch.pdf

[3] J. A. Bargas-Avila and K. Hornbæk, Old wine in new bottles or novel challenges: A critical analysis of empirical studies of user experience,", in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2689–2698. [Online]. Available: http://www.kasperhornbaek.dk/papers/CHI2011_UXReview.pdf

[4] 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, Journal of web semantics, vol. 37-38, pp. 184–206, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2016.03.003

[5] B. J. Nelson, "Remote procedure call," Ph.D. dissertation, Pittsburgh, PA, USA, 1981.

[6] E. W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math., vol. 1, no. 1, pp. 269–271, Dec. 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390

[7] R. Bellman, On a routing problem, Quarterly of Applied Mathematics, vol. 16, pp. 87–90, 1958.

[8] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, Connection Scan Algorithm, CoRR,vol. abs/1703.05997, 2017. [Online]. Available: http://arxiv.org/abs/1703.05997

[9] The iRail project, https://www.irail.be/

[10] General Transit Feed Specification, https://developers.google.com/transit/gtfs/

[11] General Transit Feed Specification Realtime, https://developers.google.com/transit/gtfs-realtime/