De door de gebruikers ervaren performantie van routeplanning-API's

Bert Marcelis

Promotoren: prof. dr. ir. Ruben Verborgh, dr. ing.Pieter Colpaert
Begeleider: Julian Andres Rojas Melendez

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. Koen De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017-2018

UNIVERSITEIT
GENT

Dankwoord

Na zes maand zwoegen is deze masterproef eindelijk klaar. Hoewel ik dit werk zelf moest maken, zou dit niet mogelijk geweest zijn zonder de hulp van een aantal mensen rondom mij.

Als eerste wil ik mijn promotor Pieter Colpaert bedanken. Pieter, zonder jouw continue begeleiding, feedback, en talloze uren aan nalezen zou ik dit werk nooit tot zo'n goed einde gebracht. Ik wil ook mijn begeleider Julian Melendez bedanken, voor de hulp bij het verder ontwikkelen van Linked Connections en het nalezen van mijn scriptie.

Daarnaast wil ik ook graag mijn ouders bedanken, voor de steun en om altijd klaar te staan als ik hulp nodig had en te zorgen dat ik na een week hard werken steeds in een warm nest kon thuiskomen. Ik wil ook mijn vriendin Sofie bedanken voor de steun, de aanmoediging en het geduld doorheen deze maanden van hard werk. Ook mijn vrienden wil ik hier niet vergeten, die steeds voor wat ontspanning konden zorgen tijdens de schrijf- of werksessies in.

Tot slot wens ik iedereen te bedanken die heeft bijgedragen aan het onderzoek, door deel te nemen aan user-testing, door de enquête in te vullen, of door mee te helpen door het verspreiden van de enquête.

Bedankt iedereen!

Bert Marcelis

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 travellers, and 81 respondents gave insights on their use of travel companions. We found that the performance of the client-side Android Linked Connections implementation 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 off-line 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, linked data, web engineering

I. Introduction

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 we finally 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 [5]. 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[1]. 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

[1]gtfs https://developers.google.com/transit/gtfs/

to application developers. It is an extension to gtfs [2], 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.

[2]gtfs-rt https://developers.google.com/transit/gtfs-realtime/

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 the client-side Android route planning implementation 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 client-side Android Linked Connections implementation the second time. After testing and grading the client-side Android Linked Connections implementation, 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[3], 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 of the hardware on the client-side Linked Connections implementation.

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

[3]The iRail project: https://hello.irail.be

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.

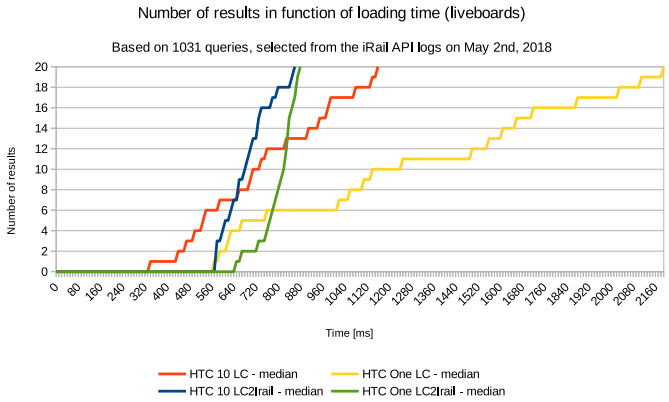


Fig. 1. The median performance of loading incremental liveboard results using different Web architectures and different devices shows that for fast devices (HTC-10 LC) client-side filtering gives the first results faster than the serverside implementation (HTC-10 LC2IRAIL). This effect is also visible, yet less apparent, for the slower device (HTC One).

The incremental results for liveboards shown in figure 1, make clear that there are 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 the Linked Connections client does not. There is a gap between the time needed by the client-side Linked Connections implementation on both devices, which grows with the number of results needed.

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 the Linked Connections client, which has a larger distribution of loading times on the HTC One. While the differences between the median response time of the Linked Connections client on the HTC 10 and lc2irail on both devices cannot be distinguished by end users, there is a clear difference with the median of the Linked Connections implementation on

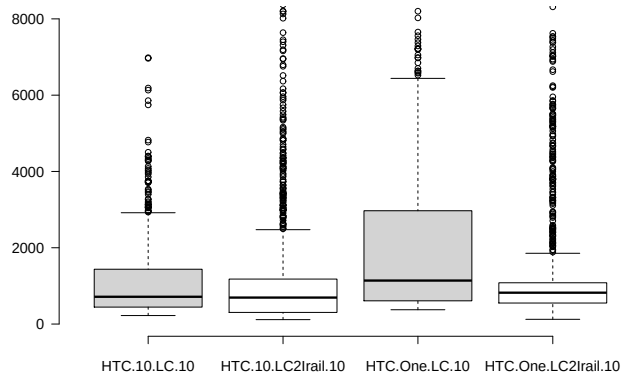


Fig. 2. The distribution of loading times for liveboard queries with 10 results, using different Web architectures and different devices. While the Linked Connections implementation on the fast device (HTC.10.LC.10) is on par with the LC2Irail performance, is the Linked Connections client performance on the HTC One less consistent.The third percentile lies 1,5 seconds higher compared to the Linked Connections client on the HTC One, meaning this implementation will give a less consistent experience on this device.

the HTC One.

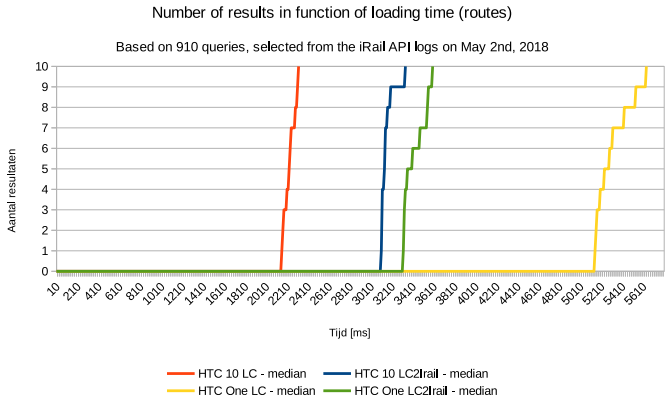


Fig. 3. The median performance of loading incremental route results using different Web architectures and different devices shows that for fast devices (HTC-10 LC) client-side filtering gives the first results faster than the server-side implementation (HTC-10 LC2IRAIL). This is opposite to slow devices, where the server-side implementation (HTC One LC2Irail) is 1,8 seconds faster than the client-side implementation (HTC One LC).

When looking at the incremental results for routes (figure 3) this difference between devices becomes even more visible. The Linked Connections client 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. The Linked Connections implementation performs better than lc2irail on the modern HTC 10, but it performs 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 the loading times of the Linked Connections client 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 in between the performance of the Linked Connections client on both devices.

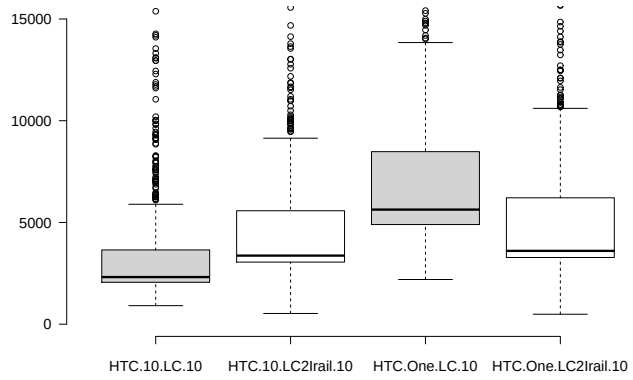


Fig. 4. The distribution of loading times for route planning queries with 10 results, using different Web architectures and different devices. The performance of LC2Irail is similar on both devices (HTC.10.LC2Irail.10 and HTC.One.LC2Irail.10), while the Linked Connections implementation performs better or worse depending on the device. On the fast device (HTC.10.LC.10), the median lies 59% (3,3s) lower compared to the slower device (HTC.One.LC.10), meaning the tenth result will be loaded more than twice as fast in 50% of the cases

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 typically needs at least three or four hours of data, which translates in at least six to eight pages, depending on the server configuration. lc2irail, which has quick access to the data, can access pages faster, and has an advantage here. Again, it performs consistently between devices, whereas the Linked Connections implementation needs two times as much time on the HTC One, compared to Linked Connections implementation on the HTC 10.

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

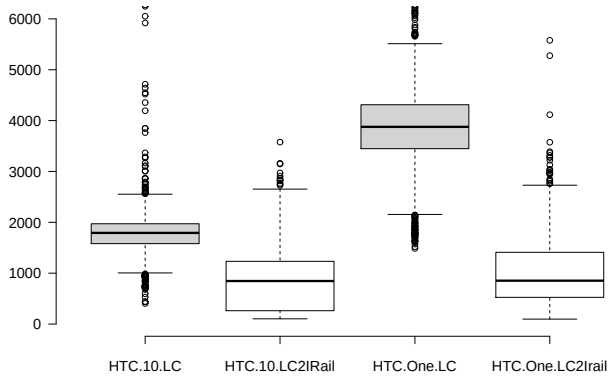


Fig. 5. The distribution of loading times for vehicle queries, using different Web architectures and different devices. The Linked Connections implementation performs worse on both devices compared to the LC2Irail performance, with the HTC One (HTC.One.LC) taking 2,16 times longer to load the same results compared to the HTC 10 (HTC.10.LC). LC2Irail performs consistent across devices, with only a 7ms difference between the median on both devices (HTC.10.LC2Irail and HTC.One.LC2Irail).

enced 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.

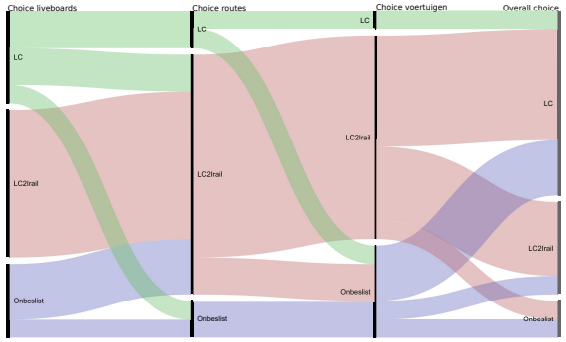


Fig. 6. The choices users make when asked to pick an implementation based on speed, and when they are asked to make a final decision based on speed and (off-line) functionality. For liveboards, routes and vehicles respectively 29%, 12% and 6% of the testers experience Linked Connections as faster, while 47%, 76% and 65% the users experience LC2Irail faster. Taking off-line access into account, 59% of the testers picks the Linked Connections implementation as their favourite.

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.json 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 various 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.

The client-side Android route planning implementation can be improved by paying attention to implementation details such as paying attention to the performance of used libraries and the use of String objects. Resulting from this research, we see evidence that only a Paged Collection of connections is not sufficient to provide for the basic functionalities of a route planning applications. For the vehicle overview for instance, we suggest adding a mandatory feature to the Linked Connections specification to publish indices on which trips are described in each page, or to allow clients to filter on a trip id. This reduces the overhead of retrieving pages without useful data.

Reducing this network traffic 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. These results based on the cached data 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 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.

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

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van listings

"De trein naar .. heeft een vermoedelijke vertraging van ... minuten. #typischvlaamsezinnetjes"

~Jan Dorie @oombom

1

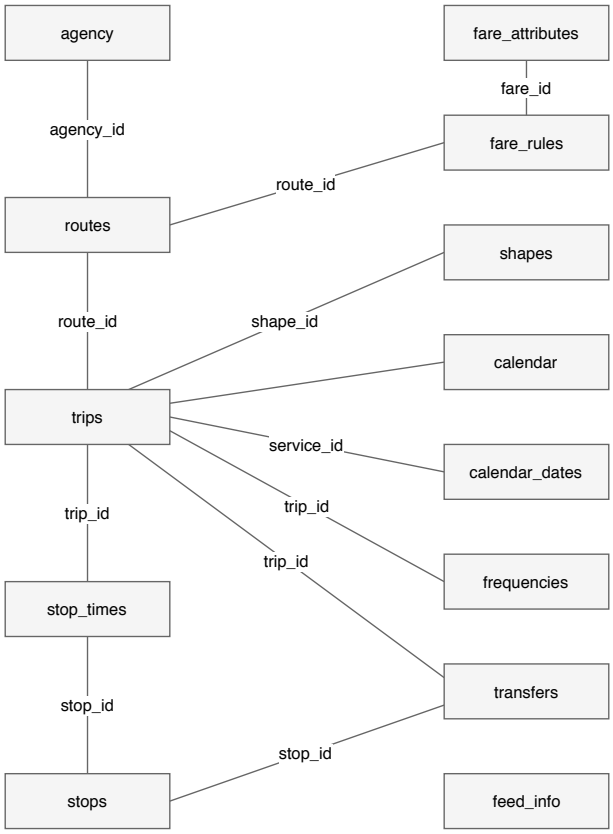
Inleiding

Openbaar vervoer is een essentiële dienst in elke stad [1]. Om vlot van dit openbaar vervoer gebruik te maken, zijn er tientallen websites en apps (user-agents of clients) die gebruikers informatie verstrekken over vertrekken, aankomsten, ritten, routes en vertragingen. Voorbeelden hiervan zijn iRail.be, HyperRail en Railer in België, en CityMapper, TheTransitApp, Here WeGo en Google maps wereldwijd. Op dit moment zijn al deze user-agents echter toegewezen op het gebruik van data dumps of specifieke API's om informatie met betrekking tot openbaar vervoer te publiceren, of een variant ervan.

Enerzijds zijn er volledige data dumps, in de vorm van General Transit Feed Specification (GTFS)[1] en General Transit Feed Specification Realtime (GTFS-RT)[2]. GTFS bestanden bevatten informatie over alle voertuigen van een dienstverlener, over een relatief grote tijdspanne, typisch enkele maanden tot een jaar. GTFS-RT-bestanden bevatten realtime informatie over ritten in de komende uren tot dagen. Om al deze data compact op te slaan en te versturen, worden deze opgeslagen als een verzameling regels in een database-achtige structuur, zoals te zien in figuur 1.1. Deze regels omschrijven wanneer welk voertuig welke rit maakt. Om op basis van deze regels vragen te beantwoorden, dient deze verzameling abstracte regels omgevormd te worden naar een gepast model waarin informatie over ritten en stopplaatsen opgevraagd kan worden, en routes berekend kunnen worden. Hiervoor zijn, afhankelijk van welke informatie ge-

[1] https://developers.google.com/transit/gtfs/

[2] https://developers.google.com/transit/gtfs-realtime/

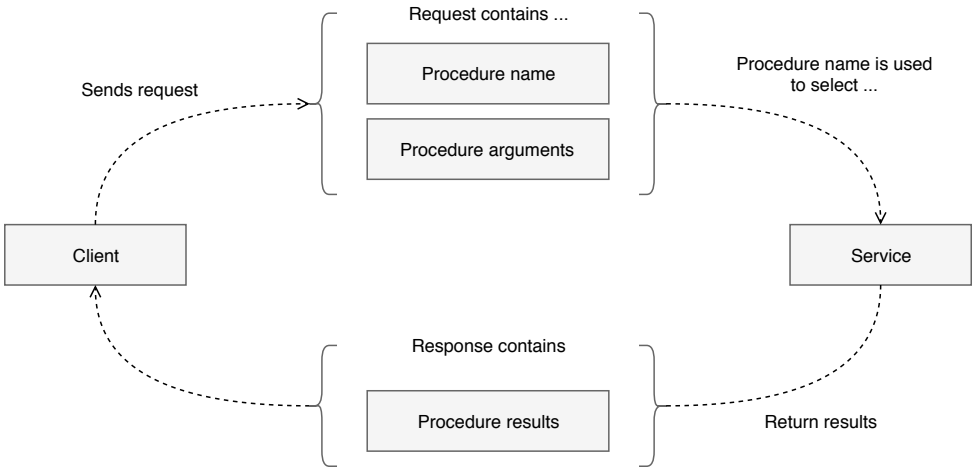


Figuur 1.1: De bestandsstructuur van GTFS data.

wenst is, zware berekeningen vereist, die afhankelijk van de grootte van het GTFS bestand vijf à tien minuten kunnen duren op een moderne computer. Gebruikers kunnen geen tien minuten wachten tot de data getransformeerd zijn, waardoor deze optie niet beschikbaar is op mobiele toestellen. Verder is dit formaat een mogelijke technologische restrictie op de reisinformatie: enkel gevorderde ontwikkelaars kunnen hiervan gebruik maken. Open data is slechts open als deze (onder andere) beschikbaar zijn in een begrijpbaar formaat [2]. GTFS is dus vooral geschikt om reisinformatie te delen met grote bedrijven, en in mindere mate voor individuele ontwikkelaars die reisinformatie eenvoudig willen visualiseren (digital signage, routeplanner applicaties, websites, ...).

Anderzijds zijn er traditionele Remote Procedure Call (RPC) API's [3] zoals iRail[3], die beschikken over verschillende endpoints om specifieke vragen te beantwoorden. Bij deze API's wordt, zoals de naam reeds doet vermoeden, een procedure op de server aangeroepen. Dit is duidelijk zichtbaar in de schematische voorstelling in figuur 1.2. Achterliggend kunnen zware berekeningen uitvoeren of grote databases geraadpleegd worden zonder dat de gebruiker hier nadeel van ondervindt. Deze antwoorden zijn rechtstreeks bruikbaar voor de client, maar bieden enkel het

[3]https://irail.be

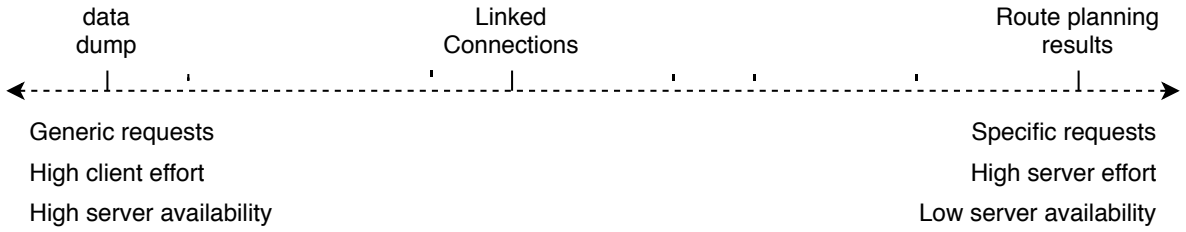


Figuur 1.2: De werkwijze van een RPC API.

antwoord op één specifieke vraag. Om een andere vraag te beantwoorden, al dan niet gesteld door dezelfde client, is een nieuw verzoek naar de server nodig, en moet het nieuwe antwoord verwerkt worden. Voor elk verzoek naar de server is een verbinding tussen de client en server benodigd, en relatief veel processortijd van de server. Een continue internetverbinding is dus vereist, en server-side is een potentieel grote en dure infrastructuur nodig om aan alle vragen te voldoen. Een ander belangrijk nadeel bij deze techniek is dat deze data moeilijk te combineren zijn met andere datasets. Een route plannen die gebruik maakt van meerdere openbaar vervoer aanbieders is enkel mogelijk als een API beschikbaar is die achterliggend door meerdere datasets zoekt. Twee API's combineren om de beste route via meerdere vervoersmaatschappijen te vinden is niet op een eenvoudige manier mogelijk.

Deze twee methodes zijn elkaars tegengestelde. Ontwikkelaars moeten kiezen voor data die compact maar complex – en slechts indirect bruikbaar – is, of voor een vraag-antwoord systeem wat voor elke nieuwe vraag een nieuw verzoek naar een server moet maken. Aan de IDLab onderzoeksgroep aan UGent is onderzoek gedaan naar Linked Connections (LC)[4], een nieuw formaat dat een nieuw evenwicht tracht te vinden tussen de bestaande oplossingen. Alle vertrekken van alle voertuigen worden in één chronologische lijst verzameld, waarbij de lijst kan opgevraagd worden volgens vaste tijdsintervallen met een grootteorde van enkele minuten. Hierdoor hoeft de server enkel deze lijst in fragmenten aan te bieden, waarbij alle clients dezelfde informatie krijgen. De clients dienen zelf nog berekeningen te maken, maar deze zijn relatief eenvoudig vergeleken met de berekeningen die nodig zijn om een GTFS feed te verwerken. Data in het Linked Connections-formaat kunnen eenvoudig toegankelijk gemaakt worden via een open-source serverapplicatie[5].

[4]https://linkedconnections.org
[5]https://github.com/julianrojas87/linked-connections-server/



Figuur 1.3: De Linked Data Fragments as illustreert dat alle HTTP-interfaces data fragmenten aanbieden, maar verschillen in hoe specifiek de aangeboden data is, en dus de moeilijkheid om deze aan te maken [4]. In deze figuur is de as toegepast op HTTP-interfaces voor routeplanning [5].

1.1 Wat is user-perceived performance?

Elke interface voor het ophalen van data heeft specifieke eigenschappen zoals latency, performance, cache-hergebruik,... [6]. Wanneer we verschillende technieken vergelijken door voor elke techniek dezelfde user-agent te gebruiken, kunnen we de impact van de verschillende achterliggende technieken op de eindgebruiker onderzoeken. Hiervoor definiëren we de *user-perceived performance*. De user-perceived performance is de performance zoals de gebruiker deze ervaart, welke niet strikt gelijk hoeft te zijn aan de werkelijke performance van technische component. De user-perceived latency werd in 2000 door Roy T. Fielding gedefinieerd als de tijd tussen het selecteren van een link en het renderen van een bruikbaar resultaat [7]. Latency treedt volgens Fielding op verschillende punten:

1. de tijd die de client nodig heeft om actie te ondernemen
2. de tijd die nodig is voor voorbereidende acties
3. de tijd om een verzoek te verzenden
4. de tijd die de server nodig heeft om het antwoord te bepalen
5. de tijd die nodig is om het antwoord te verzenden
6. de tijd om het antwoord te verwerken en weer te geven

Terwijl enkel stappen 3, 4 en 5 rechtstreeks afhankelijk zijn van het netwerk, kunnen al deze stappen beïnvloed worden door de gebruikte techniek [7].

Wanneer we ons niet enkel op de prestaties richten, maar ook op de manier waarop de gebruiker omgaat met de technologie, komen we bij de gebruikerservaring of user-experience (UX) terecht. User-experience is een brede term, die oorspronkelijk gebruikt werd voor het ontwerp en gebruik

van interfaces, waardoor het een synoniem vormde voor interacties en bruikbaarheid [8]. Een ander gebruik van de term focust op de niet-instrumentele noden en ervaringen in een complexere zin [8]. In beide gevallen is UX een centraal belang in Human-Computer Interfaces (HCI) [9]. In deze masterproef zullen we de user-experience steeds in deze bredere zin beschouwen.

Ondertussen zijn we geëvolueerd naar een wereld waarin data vaak mobiel geconsumeerd worden: 78% van de Vlamingen beschikt over een smartphone, 80,5% beschikt over een laptop. Slechts 41,8% beschikt over een desktop-computer [10]. Bij deze mobiele toestellen zijn er ook andere aspecten die meespelen in de user-experience: batterijgebruik en offline toegang tot data vormen een aanzienlijke factor [11]. Een applicatie biedt een betere ervaring wanneer deze dezelfde data kan weergeven met aanzienlijk minder energieverbruik, of wanneer deze consistent goed presteert, ook wanneer netwerk slecht of niet beschikbaar is. Hoewel de user-perceived performance een groot belang heeft in de user-experience van een applicatie, dienen we dus ook deze andere aspecten in rekening te brengen. Mobiele gebruikers hebben ook nog steeds angst om te veel data te verbruiken [12].

1.2 Probleemstelling en doel van de masterproef

Linked Connections werd ontwikkeld met de bedoeling een evenwicht te vinden tussen data dumps en RPC API's. In plaats van elke query op een server te beantwoorden, wordt een gelinkte lijst van connecties gepubliceerd. Linked Connections laat hierdoor toe om vragen te beantwoorden door middel van een lineair groeiende lijst van connecties te overlopen [5]. Bovendien gebruiken alle user-agents dezelfde lijst, waardoor deze zeer cachebaar is. Bij stijgende belasting daalt de tijd die de server nodig heeft per verzoek [13].

Terwijl de cost-efficiency van Linked Connections reeds is aangetoond, waarbij Linked Connections hetzelfde aantal verzoeken kan beantwoorden met slechts 25% van de rekencapacteit [13, 14], is er nog geen onderzoek gebeurd naar de user-perceived performance van een user-agent wanneer deze gebruik maakt van Linked Connections, vergeleken met wanneer deze zelfde user-agent gebruik maakt van een traditionele RPC API.

In deze studie richten we ons specifiek op routeplanning gebruik makend van mobiele toestellen. Deze toestellen hebben minder processorkracht en geheugen vergeleken met traditionele computers, maar ook bandbreedte en beschikbaarheid van internet zijn vaak beperkt. In het slechtste geval is er geen netwerkverbinding, waarbij enkel een cache beschikbaar is. Verder zullen we ons specifiek richten op het verschil tussen een RPC API gebaseerd op Linked Connections [13] en de originele Linked Connections webserver. Als user-agent zullen we een *fork* van de HyperRail[6] applicatie voor Android gebruiken, gemodificeerd om gebruik te maken van de genoemde

[6]https://hyperrail.be

API's. Door deze testopstelling zijn de oorspronkelijke data, de server hardware, de user-agent en de client hardware identiek bij elke vergelijking. Enkel het formaat voor serverinteracties en transport van data zal verschillen.

Om routes te berekenen zullen we gebruik maken van het Connection Scan Algorithm (CSA) [15, 16, 17]. Dit algoritme vereist een op vertrektijd gesorteerde lijst van vertrekken. Dit is de exacte definitie van de Linked Connections knowledge graph, waardoor dit algoritme zonder al te veel modificaties toegepast kan worden. Fragmenten kunnen hierbij geladen worden op het moment dat ze nodig zijn. We zullen dezelfde implementatie gebruiken zowel bij de client-side API als bij de server-side API om zo correct mogelijke resultaten te behalen.

In eerste instantie zal een traditionele (RPC) API geschreven worden welke gebruik maakt van de Linked Connections fragmenten op de Solid State Disk (SSD) van de server. Deze API zal endpoints bevatten voor het tonen van vertrekken en aankomsten per station, het berekenen van routes, en voor het weergeven van het traject per trein. Vervolgens zal een API zonder specifieke server-side geïmplementeerd worden in de applicatie. Deze zal dezelfde informatie ter beschikking stellen in de applicatie, maar zal hiervoor enkel (delen van) de gelinkte lijst met vertrekken downloaden.

Eenmaal beide API's volledig geïmplementeerd zijn, zal de user-perceived performance onderzocht worden. Hiertoe worden begeleide user tests gehouden, waarbij een aantal testers afwisselend met beide API's hun dagelijkse opzoekingen zullen uitvoeren, waarna ze aan de hand van een vragenlijst bevraagd zullen worden naar hun ervaringen en voorkeuren. Het is essentieel om de subjectieve ervaringen van gebruikers te bevragen, gezien verschillende gebruikers mogelijk verschillende afwegingen maken. We verwachten dat sommige gebruikers offline toegang waardevol zullen vinden, terwijl anderen mogelijk geen belang hechten aan offline toegang. Ook zal getracht worden technische data te verzamelen, zoals geheugen- en processorgebruik, laadtijden, en batterijverbruik.

Deze masterproef zal gebruik maken van data afkomstig van de NMBS om routeplanning en realtime data over treinen in België weer te geven. Door de bron van de data te vervangen kan dit onderzoek ook toegepast worden op andere openbaar vervoersmaatschappijen die gebruik maken van tijdsschema's, ongeacht het soort voertuig dat gebruikt wordt.

1.3 Onderzoeksvraag

Zoals in de vorige sectie werd besproken, zullen we onderzoeken welke API de beste gebruikerservaring oplevert. Dit formuleren we nu in een onderzoeksvraag, die we zullen beantwoorden in hoofdstuk 5.

Onderzoeksvraag Verbetert de user-experience en user-perceived performance van een applicatie voor openbaar vervoer wanneer gebruik gemaakt wordt van Linked Connections in plaats van traditionele RPC API's?

Regelmatig klagen mensen over de kwaliteit van hun netwerkverbinding bij treinreizen. Hierbij klaagt men over een trage of afwezige verbinding. We vermoeden dat de gebruiker hierdoor gehinderd wordt bij het online opzoeken van informatie, en liefst op elk moment van zijn treinreis over reisinformatie wilt kunnen beschikken, ongeacht de kwaliteit van het mobiele netwerk.

Hypothese 1 Gebruikers beschikken niet over een kwalitatieve internetverbinding tijdens het reizen, en ervaren de mogelijkheid voor offline zoekopdrachten als een meerwaarde.

Uit gesprekken met treinreizigers blijkt dat een aantal reizigers snellere routes kent dan de routes die applicaties op basis van officiële data voorstellen. Zo kan een lange overstap worden voorkomen door zich te haasten bij het overstappen. We vermoeden dat gebruikers het handig vinden om deze routes ook in hun routeplanning applicatie te kunnen terugvinden. We vermoeden ook dat gebruikers met een beperkte mobiliteit graag beperkingen zouden kunnen stellen aan de route, om bijvoorbeeld niet over te stappen in stations zonder verhoogde perrons.

Hypothese 2 De gebruiker ervaart de mogelijkheid om voorkeuren voor routes in te stellen (overstaptijd, toegankelijkheid, ...) als een meerwaarde.

Zoals eerder vermeld hebben mobiele gebruikers ook nog steeds angst om te veel data te verbruiken [12]. We vermoeden dat dit ook nog steeds zo is, maar dit geen zorg is van gebruikers wanneer ze applicaties voor openbaar vervoer gebruiken.

Hypothese 3 De gebruiker heeft angst om te veel mobiele data te verbruiken, maar let hier niet op bij het gebruik van routeplanning-apps.

Hoewel privacy een *hot topic* is, vermoeden we dat de modale treinreiziger geen belang hecht aan zijn of haar privacy bij het gebruik van routeplanning applicaties. We vermoeden dat de gebruiker extra privacy als positief ervaart, maar zijn keuze voor een routeplanning applicatie hier niet door laat beïnvloeden.

Hypothese 4 De gebruiker ervaart extra privacy bij het opzoeken van routes niet als een noemenswaardige meerwaarde

"Optimization before measurement is the root of all evil"

~D. Knuth

2

Implementatie

Om een zo eerlijk mogelijke vergelijking te bekomen, zullen we zowel bij de client-side implementatie van Linked Connections als de server-side API (LC2Irail) dezelfde algoritmes toepassen. Gezien de jonge leeftijd van het Linked Connections framework zijn er nog geen algoritmes beschikbaar om deze data te verwerken. We zullen de ontwikkeling van deze algoritmen bespreken, met speciale aandacht voor het routeplanning algoritme vanwege de hogere complexiteit en de uitgebreide mogelijkheden.

2.1 Linked Connections specificaties

Zoals eerder aangehaald publiceert Linked Connections geen dump van reisinformatie of een volledige routeplanner, maar een lijst van zogenoemde connecties. Een connectie is het kleinste ondeelbaar stuk van een treinrit, en beschrijft het vertrek in een station en de aankomst in het volgende station op de route. Deze connecties worden opgeslagen in een lijst en gesorteerd volgens vertrektijd. Hierna wordt deze lijst gesplitst, om pagina's van gelijke grootte of gelijke tijdsduur te bekomen. Deze fragmenten kunnen gepubliceerd worden via HTTP als *JSON-LD*[1], waarbij user-agents kunnen kiezen welke pagina's ze opvragen. Links in de gepubliceerde documenten zorgen ervoor dat user-agents steeds weten welke pagina ze als volgende moeten laden [18].

[1] https://json-ld.org/

Om bovenstaande methode in de praktijk om te zetten, wordt gebruik gemaakt van de open source LC-Server[2]. Om GTFS om te zetten naar Linked Connections, wordt er achterliggend gebruik gemaakt van de gtfs2lc tool[3].

Deze data zijn publiek toegankelijk via https://graph.irail.be/.

2.1.1 Vraag- en antwoordformaat

Om een pagina met data op te halen, wordt een verzoek gemaakt naar de API, waarbij de vervoersmaatschappij en het gewenste tijdstip in ISO8601 formaat in de URL opgenomen worden. In codefragmenten 2.1 en 2.2 zien we het resultaat voor de NMBS op 20 maart 2018, 12:30. Het verzoek dat gemaakt werd om de resultaten te bekomen is een HTTP verzoek naar `https://graph.irail.be/sncb/connections?departureTime=2018-03-20T12:30:00.000Z`.

We overlopen nu de belangrijkste velden in dit antwoord:

1. *@context*: Deze lijst, zichtbaar in codefragment 2.1, definieert de gebruikte namespaces en velden
2. *hydra:next* en *hydra:previous*: Links naar de pagina met respectievelijk de volgende en de voorgaande data
3. *hydra:search*: Informatie over de huidige pagina
4. *@graph*: Deze lijst, zichtbaar in codefragment 2.2, bevat de eigenlijke data. Elk vertrek bevat de volgende informatie:
   (a) *departureStop*: De URI welke het station van vertrek uniek identificeert.
   (b) *arrivalStop*: De URI welke het station van aankomst uniek identificeert.
   (c) departureTime, arrivalTime: De geplande tijden, respectievelijk bij vertrek en aankomst.
   (d) *departureDelay*, *arrivalDelay*: De vertraging, respectievelijk bij vertrek en aankomst.
   (e) *direction*: De richting van dit voertuig, wat vaak ook op de lichtkrant van het voertuig weergegeven wordt.
   (f) *gtfs:trip*: Een URI welk de rit van het voertuig uniek identificeert
   (g) *gtfs:route*: Een URI welk de route van het voertuig uniek identificeert
   (h) *gtfs:pickupType* en *gtfs:dropOffType*: geeft aan of reizigers al dan niet kunnen op- of afstappen bij respectievelijk vertrek en aankomst

```
{"@context": {
    "lc": "http://semweb.mmlab.be/ns/linkedconnections#",
    "hydra": "http://www.w3.org/ns/hydra/core#",
    "gtfs": "http://vocab.gtfs.org/terms#",
    "...": "..."
},
"@id": "https://graph.irail.be/sncb/connections?departureTime=2018-03-20T12:30:00.000Z",
"@type": "hydra:PagedCollection",
"hydra:next":
↪  "https://graph.irail.be/sncb/connections?departureTime=2018-03-20T12:40:00.000Z",
"hydra:previous":
↪  "https://graph.irail.be/sncb/connections?departureTime=2018-03-20T12:20:00.000Z",
"hydra:search": {"...": "..."},
"...":"..."}
```

Code 2.1: Voorbeeld Linked Connections formaat: context

```
{"...":"...",
"@graph": [
  {   "@id": "http://irail.be/connections/8822228/20180320/S11961",
      "@type": "Connection",
      "departureStop": "http://irail.be/stations/NMBS/008822228",
      "arrivalStop": "http://irail.be/stations/NMBS/008822210",
      "departureTime": "2018-03-20T12:30:00.000Z",
      "departureDelay": 60,
      "arrivalTime": "2018-03-20T12:32:00.000Z",
      "arrivalDelay": 0,
      "direction": "Anvers-Central",
      "gtfs:trip": "http://irail.be/vehicle/S11961/20180320",
      "gtfs:route": "http://irail.be/vehicle/S11961",
      "gtfs:pickupType": "gtfs:Regular",
      "gtfs:dropOffType": "gtfs:Regular"
  },
  {"...":"...", }
]}
```

Code 2.2: Voorbeeld Linked Connections formaat: graph

De volledige specificatie kan teruggevonden worden op de LC website[4].

[2]https://github.com/julianrojas87/linked-connections-server/
[3]https://github.com/linkedconnections/gtfs2lc
[4]https://linkedconnections.org/specification/1-0

2.1.2 Open World Assumption

Een van de speerpunten van Linked Connections is het gemak waarmee data gecombineerd kunnen worden. Om data van meerdere vervoersmaatschappijen te combineren, hoeft men enkel de Linked Connections voor deze vervoersmaatschappijen te downloaden, en *merge-sort* toepassen. Wanneer het niet mogelijk is een route tussen twee punten te plannen, is het ook nog steeds mogelijk dat deze route gepland kan worden wanneer data van een extra vervoersmaatschappij toegevoegd wordt. Dit heet de Open World Assumption. Formeel stellen we dat de open world assumption gedefinieerd wordt door de veronderstelling dat de "waarheid"van een statement of hypothese niet afhangt van het al dan niet bekend zijn van deze hypothese door een enkele agent [19]. Met andere woorden is mogelijk dat wanneer een client een vraag niet kan beantwoorden, het nog steeds mogelijk is dat er een antwoord op deze vraag is wanneer meer data toegevoegd wordt.

2.2 Gebruikte systemen doorheen dit onderzoek

In het kader van dit onderzoek zullen we twee implementaties ontwikkelen. Enerzijds ontwikkelen we een client-side implementatie van Linked Connections, waarbij data opgehaald wordt van de bestaande Linked-Connections-Server. Anderzijds ontwikkelen we ook een RPC API die als referentie zal dienen, LC2Irail. Deze API hanteert dezelfde algoritmes als de lokale implementatie.

Linked Connections Server Deze term doelt op de bestaande NodeJS applicatie welke Linked Connections fragmenten beheert op een server, en deze over HTTP aanbiedt. Beide API implementaties maken gebruik van deze fragmenten. Broncode is beschikbaar op https://github.com/julianrojas87/linked-connections-server/.

Linked Connections, LC, lokale implementatie Deze termen doelen op de implementatie waarbij data van de Linked-Connections-Server opgehaald wordt en op het mobiele toestel verwerkt wordt. Broncode is beschikbaar op https://github.com/Bertware/masterthesis-LC-LC2Irail-android-client.

LC2Irail, server implementatie Deze termen doelen op de implementatie van een RPC API waarbij data in de vorm van Linked Connections fragments van de harde schijf opgehaald wordt, om daarna op de server verwerkt te worden en het antwoord naar de mobiele client te sturen. Broncode is beschikbaar op https://github.com/Bertware/masterthesis-lc2Irail.

Beide implementaties beschikken elk over dezelfde drie types opzoeking.

Liveboards Deze opzoeking geeft de gebruiker alle vertrekkende of aankomende treinen voor

een bepaald station. Er zijn theoretisch gezien oneindig veel resultaten beschikbaar door het beschouwde tijdsinterval steeds uit te breiden.

Routes Deze opzoeking geeft de gebruiker routes van A naar B, waarbij stations A en B door de gebruiker zijn gekozen. Er zijn theoretisch gezien oneindig veel resultaten beschikbaar door het beschouwde tijdsinterval steeds uit te breiden.

Voertuigen Deze opzoeking geeft het volledige traject van een voertuig weer. Het volledig traject wordt als één resultaat beschouwd.

2.3 Algoritmes

2.3.1 Connection Scan Algoritme

Veelgebruikte algoritmes voor routeplanning zijn (varianten op) het algoritme van Dijkstra [20, 15, 21, 22] of Bellman-Ford [23]. Toepassingen die gebruik maken van deze algoritmes vereisen echter het bijhouden van een graaf, en in het geval van Dijkstra ook een *priority queue*. Naast de impact op prestaties die deze eisen vormen, beperkt een graaf ook de flexibiliteit. De *Open World Assumption* stelt dat er steeds andere stopplaatsen zijn wiens bestaan we (nog) niet kennen. Het opstellen van een graaf zou vereisen dat we alle gegevens eerst volledig moeten downloaden, terwijl Linked Connections net goed geschikt is voor streaming.

Het Connection Scan Algoritme (CSA) werd voor het eerst beschreven door Ben Strasser in 2013 [15]. Dit algoritme vereist een lijst met vertrekken gesorteerd op vertrektijd. Hiermee worden alle routes in een tijdsinterval efficiënt berekend [16, 17]. In tegenstelling tot Dijkstra's algoritme is er geen graaf of *priority queue* nodig. Waar andere algoritmen ofwel enkel op kleine netwerken performant zijn, ofwel niet altijd de best mogelijke route vinden, kan CSA de optimale route in grote netwerken toch efficiënt vinden [16].

In de praktijk blijkt de standaard implementatie van CSA snel genoeg voor realtime opzoekingen, ook voor grote spoornetwerken zoals het Duitse net [16]. Wanneer Linked Connections op grote schaal toegepast wordt, kan ook de performantie van CSA nog verder verbeterd worden door het implementeren van Connection Scan Accelerated [16, 17]. Een andere optie is het toepassen van heuristieken. Deze verbeteren wel de responstijd, maar zorgen er in sommige gevallen voor dat een optimale route niet gevonden wordt [21]. Gezien het Belgische spoornetwerk relatief klein is in vergelijking met andere landen, zullen we hier rechtstreeks CSA op toepassen zonder van verdere heuristieken gebruik te maken.

Wanneer we een routeplanning algoritme willen gebruiken voor een spoornetwerk, is het vooral

belangrijk om snel rekening te kunnen houden met vertragingen bij vertrek of aankomst [16, 17]. Als een trein met vertraging zal vertrekken, hoeft enkel de connectie die bij dit vertrek hoort aangepast te worden. Hierna zal CSA weer correcte resultaten geven rekening houdend met de vertraging. Met andere woorden hoeven we dus gewoon de Linked Connections-pagina opnieuw te laden om een nieuwe lijst te verkrijgen met de huidige vertragingen.

Net zoals Dijkstra en andere algoritmes berekent CSA de snelste route, al is dit duidelijk niet altijd de route die de gebruiker wenst. Zo kan de snelste route nog steeds een station meermaals bezoeken, of kan men van een trein afstappen om later op deze zelfde trein weer op te stappen [16]. Terwijl intuïtief duidelijk is dat deze routes niet de beste routes zijn, kan de reistijd bij deze trajecten even lang zijn als de snelste route. Het algoritme kan dergelijke routes dus als antwoord kiezen.

Een oplossing hiervoor is om ervoor te zorgen dat de resultaten pareto-optimaal zijn. Een resultaat is pareto-optimaal als het niet gedomineerd wordt door een ander resultaat. Een resultaat q wordt gedomineerd door een ander resultaat p als p voor minstens één criterium een beter waarde heeft, en voor geen enkel criterium een slechtere waarde heeft dan q [22, 17]. Door ook het aantal overstappen te optimaliseren voorkomen we dat van eenzelfde trein wordt afgestapt om later terug op te stappen [16]. Een oplossing met een overstap te veel heeft immers een criterium met een slechtere waarde, en zal dus nooit pareto-optimaal zijn.

Naast de tijd van aankomst zijn er nog een aantal andere criteria die vaak geoptimaliseerd worden. Na reistijd is het aantal overstappen het tweede populairste criterium, gevolgd door de prijs van de reis [17]. Optimalisatie van de prijs is echter zeer complex vanwege de complexe tariefplannen bij openbaar vervoer [24]. Dit valt buiten de context van deze masterproef.

De werking en implementatie van CSA worden uitvoerig behandeld in [17]. De implementatie en evolutie in dit project zal worden uitgelegd aan de hand van code fragmenten in Java. Er wordt hierbij verondersteld dat sectie 4.2 in [17] gekend is. De code is asynchroon, waarbij na het laden van de eerste Linked Connections-pagina een *callback* functie opgeroepen wordt om deze pagina te verwerken. Afhankelijk van het resultaat van deze verwerking, wordt er een nieuwe pagina opgevraagd, of wordt het resultaat doorgegeven aan de oproepende code door middel van callbacks. De volledige broncode van de implementatie is online beschikbaar, zowel in Java[5] als PHP[6].

Wanneer CSA geïmplementeerd wordt merken we duidelijke verschillen met de voorgestelde routes door de NMBS. Deze verschillen manifesteren zich vooral in de keuze van het station

[5] `https://github.com/Bertware/linkedconnections-android-client/blob/master/Hyperrail/src/main/java/be/hyperrail/android/irail/implementation/linkedconnections/RouteResponseListener.java`

[6] `https://github.com/hyperrail/lc2irail/blob/master/app/Http/Repositories/ConnectionsRepository.php`

waar er overgestapt moet worden tussen twee treinen, en de keuze van de tussenliggende treinen indien er meer dan één overstap is. Zo is het mogelijk dat er wordt aangeraden om een trein te nemen langs Brussel-Zuid en Brussel-Centraal tot Brussel-Noord, om van daar een andere trein te nemen die op zijn beurt van Brussel-Noord langs Brussel-Centraal naar Brussel-Zuid rijdt. Verder zullen we ook nog aanpassingen doorvoeren om eenvoudig het aantal overstappen te beperken, en om op een meer uitgebreide manier aan *journey-extraction* te doen.

```
class TrainProfile {
  /**
  * The arrival time at the final destination
  */
  DateTime arrivalTime;

  /**
  * The number of transfers until the destination when hopping on to this train
  */
  int transfers;

  /**
  * The arrival connection for the next transfer or arrival
  */
  LinkedConnection arrivalConnection;
}
```

Code 2.3: In tegenstelling tot [17] wordt niet enkel de aankomsttijd, maar ook de afstaphalte en het aantal overstappen bijgehouden per trip.

Allereerst dienen we twee wijzigingen door te voeren aan de gegevensstructuren. De arrays S en T, waarin respectievelijk zogeheten stopprofielen en aankomsttijden bijgehouden werden, zijn vervangen door een Map, waardoor we onbeperkt nieuwe stations en trips kunnen toevoegen, en deze kunnen opvragen op basis van hun URI. Dit maakt het algoritme geschikt om te werken rekening houdend met de open world assumption. In de datastructuur voor een voertuig (fragment 2.3) houden we niet enkel bij wanneer we zouden aankomen, maar ook met hoeveel overstappen (beginnend na het opstappen op deze trein) we zouden aankomen, en waar we moeten afstappen van deze trein. Dit laatste is essentieel om niet enkel de aankomsttijd, maar ook de exacte route met alle overstappen te kunnen weergeven.

Ook de paren van vertrek- en aankomsttijd per station, zogenoemde profielen, worden vervangen door een meer uitgebreide gegevensstructuur, zichtbaar in fragment 2.4. Naast de vertrek- en aankomsttijd houden we nu ook de connectie bij waarmee we vertrekken in dit station op dit tijdstip, en de connectie waarmee we aankomen in het volgend station waar we moeten over- of afstappen. Ook het aantal overstappen, beginnend met tellen na het opstappen in dit station, wordt bijgehouden.

Wanneer we de ingeladen connecties willen verwerken, filteren we alle connecties uit de pagina

```
  class StationStopProfile {
  /**
  * The departure time in this stop
  */
  DateTime departureTime;

  /**
  * The arrival time at the final destination
  */
  DateTime arrivalTime;

  /**
  * The departure connection in this stop
  */
  LinkedConnection departureConnection;

  /**
  * The arrival connection for the next transfer or arrival
  */
  LinkedConnection arrivalConnection;

  /**
  * The number of transfers between standing in this station and the destination
  */
  int transfers;
}
```

Code 2.4: In tegenstelling tot [17] wordt niet enkel de vertrek- en aankomsttijd, maar ook het aantal overstappen en de afstaphalte van de volgende trein bijgehouden.

die ofwel te vroeg, ofwel te laat vallen. Zoals te zien in fragment 2.5 stellen we een *flag* in wanneer we voorbij de vroegst toegelaten vertrekdatum zijn. In dit geval zullen we na het overlopen van deze lijst geen nieuwe lijsten meer ophalen.

Het bepalen van de aankomsttijd bij wandelen, T1, en de aankomsttijd bij het gezeten blijven in de trein, T2, loopt vrijwel gelijk aan de implementatie uit [17]. Wandelen van een station naar het eindstation wordt niet ondersteund in onze implementatie. Dit implementeren is relatief eenvoudig door het gebruiken van inter-stop foothpaths [17, 22], maar deze data is op dit moment niet beschikbaar, en het verzamelen, opschonen en valideren van deze data valt buiten de context van deze masterproef. In fragment 2.6 zien we hoe het aantal overstappen wordt bepaald. In het geval dat er geen aankomst mogelijk is (binnen de beperkte tijd) stellen we zowel de aankomsttijd als het aantal overstappen in op een onrealistisch hoog getal. Dit vereenvoudigt de code aanzienlijk, aangezien er geen rekening gehouden hoeft te worden met het mogelijk leeg zijn van variabelen.

Bij de bepaling van T3, terug te vinden in fragment 2.7, maken we de eerste grote afwijking van het oorspronkelijk algoritme. Om te bepalen of een overstap mogelijk is, moeten er reeds profielen voor dit station bekend zijn. Indien dit het geval is, gaan we op zoek naar het profiel waarbij er

```
if (data.connections.length == 0) {
  mLinkedConnectionsProvider.getLinkedConnectionByUrl(data.previous, this, this, null);
  return;
}

boolean hasPassedDepartureLimit = false;
for (int i = data.connections.length - 1; i >= 0; i--) {
  LinkedConnection connection = data.connections[i];

  if (connection.departureTime.isAfter(mArrivalLimit)) {
    continue;
  }
  if (connection.departureTime.isBefore(mDepartureLimit)) {
    hasPassedDepartureLimit = true;
    continue;
  }

  ...
}
```

Code 2.5: Connecties worden overlopen volgens dalende vertrektijd. Er worden beperkingen gesteld op vertrek- en aankomsttijd.

genoeg tijd is om over te stappen, maar waarbij het aantal overstappen het maximum aantal niet overschrijdt. De intra-footpaths [17, 22], nodig voor het bepalen van de tijd die reiziger nodig heeft om over te stappen in een station, worden ingesteld op een standaardwaarde die eventueel door de gebruiker gewijzigd kan worden. Dit is nodig aangezien tijdens de ontwikkeling nog geen officiële waarden gepubliceerd waren[7]. Wanneer we een overstap vinden die aan deze voorwaarden voldoet, verhogen we het aantal overstappen ook met één. Deze aanpak is eenvoudiger dan de array-gebaseerde aanpak omschreven in [17]. Het voordeel van deze aanpak is dat automatisch alle snelste opties worden bijgehouden, zolang hun aantal overstappen onder het maximum blijft.

In plaats van de door [17] voorgestelde verhoging van de aankomsttijd met één, om zo routes met een gelijke aankomsttijd maar minder overstappen voorkeur te geven, verhogen we hier de aankomsttijd met een vooraf gedefinieerd aantal seconden. Dit aantal geeft aan hoeveel seconden we langer op een trein wensen te zitten, in plaats van over te stappen. Door dit in te stellen op 240, wordt aangegeven dat een route die er tot 4 minuten langer over doet, met een overstap minder, toch de voorkeur krijgt over de snellere route met meer overstappen. Dit is een eerste veld dat eventueel door gebruikers ingesteld zou kunnen worden om de routes te personaliseren.

Bij het bepalen van de vroegste aankomsttijd (fragment 2.7), wordt nu ook het aantal overstappen dat bij deze aankomsttijd hoort bepaald, en de connectie waar van de trein afgestapt wordt. Wanneer de aankomsttijd met overstap gelijk is aan de aankomsttijd bij blijven zitten, kiezen we voor de aankomsttijd met overstap. Dit kan contra-intuïtief lijken, maar we moeten hierbij onthouden dat bij aankomsttijd T3 reeds een constante ‘straftijd‘ is opgeteld (in dit geval 240

[7]Sinds april is deze data beschikbaar via iRail: `https://github.com/iRail/stations/pull/127`

```
 if (Objects.equals(connection.arrivalStationUri,
 ↪  mRoutesRequest.getDestination().getSemanticId())) {
  T1_walkingArrivalTime = connection.arrivalTime;
  T1_transfers = 0;
} else {
  T1_walkingArrivalTime = infinite;
  T1_transfers = 999;
}

// Determine T2, the first possible time of arrival when remaining seated
if (T.containsKey(connection.trip)) {
  T2_stayOnTripArrivalTime = T.get(connection.trip).arrivalTime;
  T2_transfers = T.get(connection.trip).transfers;
} else {
  T2_stayOnTripArrivalTime = infinite;
  T2_transfers = 999;
}
```

Code 2.6: Het aantal overstappen wordt bepaald bij het bepalen van minimale aankomsttijden

seconden). Hierdoor zullen deze aankomsttijden enkel gelijk zijn indien T2 even veel overstappen bevat. In dat geval verkiezen we om 'nu' over te stappen, waarbij 'nu' vroeger ligt dan T2: we overlopen alle vertrekken immers van laatst naar vroegst.

Door de extra toevoegingen voor journey extraction en het optimaliseren van de routes, is het bijwerken van de gegevenstructuren aanzienlijk ingewikkelder vergeleken met de originele implementatie. Voor voertuigen houden we niet langer enkel de aankomsttijd, maar ook de afstaphalte bij. Hierbij verkiezen we de halte waarlangs we zo snel mogelijk aankomen, maar bij gelijke aankomsttijd wensen we een zo lang mogelijke periode voor de overstap. Wanneer de aankomsttijd gelijk is, onderzoeken we of de nieuwe afstaphalte (de connectie die op dit moment onderzocht wordt) meer tijd voor een overstap geeft. Indien dit het geval is, werken we de afstaphalte bij. Het bijwerken van een bestaande trip is zichtbaar in fragment 2.9.

Het bijwerken van de stopprofielen, zichtbaar in fragment 2.10, is lichtjes aangepast om de efficiëntie te verhogen. De vroegste vertrekken worden nu achteraan toegevoegd. Door deze aanpassing, en het gegeven dat de vertrektijd van de huidige connectie altijd[8] gelijk of kleiner dan de vertrektijd van alle vorige connecties is, hoeven we nu enkel het laatste profiel in de lijst te evalueren.

Als de vertrektijd kleiner of gelijk is, moet de aankomsttijd kleiner zijn. we controleren dus enkel of de aankomsttijd kleiner is, en zo ja, of de vertrektijd kleiner of gelijk is. Afhankelijk van deze laatste controle voegen we een nieuw item toe aan de lijst, of vervangen we het laatste. Aangezien we telkens enkel toevoegen wanneer de aankomsttijd vroeger ligt, zal deze lijst altijd gesorteerd zijn volgens dalende aankomsttijd. Hiermee is bewezen dat deze optimalisatie correct

[8] We overlopen de lijst immers volgens niet-stijgende vertrektijd

```
if (S.containsKey(connection.arrivalStationUri)) {
  int position = S.get(connection.getArrivalStationUri()).size() - 1;
  StationStopProfile stopProfile = S.get(connection.getArrivalStationUri()).get(position);

  while ((stopProfile.departureTime.getMillis() - 300 * 1000 <=
  ↪  connection.getArrivalTime().getMillis() ||
  stopProfile.transfers >= maxTransfers) && position > 0) {
    position--;
    stopProfile = S.get(connection.getArrivalStationUri()).get(position);
  }
  if (stopProfile.departureTime.getMillis() - 300 * 1000 >
  ↪  connection.getArrivalTime().getMillis() && stopProfile.transfers <= maxTransfers) {
    T3_transferArrivalTime = new DateTime(stopProfile.arrivalTime.getMillis() + 240 * 1000);
    T3_transfers = stopProfile.transfers + 1;
  } else {
    T3_transferArrivalTime = infinite;
    T3_transfers = 999;
  }
} else {
  T3_transferArrivalTime = infinite;
  T3_transfers = 999;
}
```

Code 2.7: Bij een eventuele overstap worden ook extra factoren in rekeningen gebracht.

is, en een beter alternatief voor het overlopen van de volledige lijst.

De lijst met volledige routes reconstrueren (fragment 2.11) is relatief eenvoudig. Voor elk profiel horend bij de locatie van waar de reiziger vertrekt, volgen we de vertrek- en aankomstconnecties. Om bij elke tussenstop de juiste connectie te vinden waarmee de reis verder zal gezet worden, vergelijken we de aankomsttijd uit het stopprofiel waaruit we vertrokken, met de aankomsttijden uit de stopprofielen van de tussenstop (fragment 2.12). Wanneer deze gelijk zijn, hebben we het volgende deel van de reis gevonden. Hierbij is het belangrijk dat de aankomsttijd ook ´straftijd´ voor eventuele overstappen bevat. Deze wordt dan ook enkel intern gebruikt - zodra aan journey extraction gedaan wordt en informatie voor de gebruiker verzameld wordt, gebruiken we de vertrek- en aankomsttijden die omschreven worden in connecties.

```
if (T3_transferArrivalTime.getMillis() <= T2_stayOnTripArrivalTime.getMillis()) {
  Tmin = T3_transferArrivalTime;
  exitTrainConnection = connection;
  numberOfTransfers = T3_transfers;
} else {
  Tmin = T2_stayOnTripArrivalTime;
  if (T2_stayOnTripArrivalTime.isBefore(infinite)) {
    exitTrainConnection = T.get(connection.trip).arrivalConnection;
  } else {
    exitTrainConnection = null;
  }
  numberOfTransfers = T2_transfers;
}

// For equal times, we prefer just arriving.
if (T1_walkingArrivalTime.getMillis() <= Tmin.getMillis()) {
  Tmin = T1_walkingArrivalTime;
  exitTrainConnection = connection;
  numberOfTransfers = T1_transfers;
}

if (Tmin.isEqual(infinite)) {
  continue;
}
```

Code 2.8: Bepalen van de vroegste aankomsttijd

```
    if (Tmin.isEqual(T.get(connection.getTrip()).arrivalTime)
  &&
  ↪  !T.get(connection.getTrip()).arrivalConnection.getArrivalStationUri().equals(mRoutesRequest.getDestina
  && T3_transferArrivalTime.isEqual(T2_stayOnTripArrivalTime)
  && S.containsKey(T.get(connection.getTrip()).arrivalConnection.getArrivalStationUri())
  && S.containsKey(connection.getArrivalStationUri())
) {
  LinkedConnection currentTrainExit = T.get(connection.getTrip()).arrivalConnection;

  StationStopProfile stationStopProfile = new StationStopProfile();
  stationStopProfile.departureTime = connection.getDepartureTime();
  stationStopProfile.departureConnection = connection;

  stationStopProfile.arrivalTime = Tmin;
  stationStopProfile.arrivalConnection = currentTrainExit;

  Duration currentTransfer = new Duration(currentTrainExit.getArrivalTime(),
  ↪  getFirstReachableConnection(stationStopProfile).departureTime);

  // New situation
  stationStopProfile.arrivalTime = Tmin;
  stationStopProfile.arrivalConnection = exitTrainConnection;
  Duration newTransfer = new Duration(exitTrainConnection.getArrivalTime(),
  ↪  getFirstReachableConnection(stationStopProfile).departureTime);

  // If the new situation is better
  if (newTransfer.isLongerThan(currentTransfer)) {
    TrainProfile trainProfile = new TrainProfile();
    trainProfile.arrivalTime = Tmin;
    trainProfile.arrivalConnection = exitTrainConnection;
    trainProfile.transfers = numberOfTransfers;

    T.put(connection.getTrip(), trainProfile);
  }
}

if (Tmin.isBefore(T.get(connection.getTrip()).arrivalTime)) {
  TrainProfile trainProfile = new TrainProfile();
  trainProfile.arrivalTime = Tmin;
  trainProfile.arrivalConnection = exitTrainConnection;
  trainProfile.transfers = numberOfTransfers;

  T.put(connection.getTrip(), trainProfile);
}
```

Code 2.9: Bijwerken van de trips gegevensstructuur.

```
StationStopProfile newProfile = new StationStopProfile();
newProfile.departureTime = connection.getDepartureTime();
newProfile.arrivalTime = Tmin;
newProfile.departureConnection = connection;
newProfile.arrivalConnection = T.get(connection.getTrip()).arrivalConnection;
newProfile.transfers = numberOfTransfers;
if (S.containsKey(connection.getDepartureStationUri())) {
  int numberOfPairs = S.get(connection.getDepartureStationUri()).size();
  StationStopProfile existingProfile =
  ↪  S.get(connection.getDepartureStationUri()).get(numberOfPairs - 1);

  if (newProfile.arrivalTime.isBefore(existingProfile.arrivalTime)) {
    if (newProfile.departureTime.isEqual(existingProfile.departureTime)) {
      S.get(connection.getDepartureStationUri()).remove(numberOfPairs - 1);
      S.get(connection.getDepartureStationUri()).add(numberOfPairs - 1, newProfile);
    } else {
      S.get(connection.getDepartureStationUri()).add(newProfile);
    }
  }
} else {
  S.put(connection.getDepartureStationUri(), new ArrayList<StationStopProfile>());
  S.get(connection.getDepartureStationUri()).add(newProfile);
}
```

Code 2.10: Bijwerken van de stops gegevensstructuur.

```
// Results? Return data
Route[] routes = new Route[S.get(mRoutesRequest.getOrigin().getSemanticId()).size()];

int i = 0;
for (StationQuintuple quint : S.get(mRoutesRequest.getOrigin().getSemanticId())
) {
  // it will iterate over all legs
  StationQuintuple it = quint;
  List<RouteLeg> legs = new ArrayList<>();

  while (!Objects.equals(it.arrivalConnection.arrivalStationUri,
  ↪  mRoutesRequest.getDestination().getSemanticId())) {
    // use it.departureConnection and it.arrivalConnection to construct legs of this journey
    legs.add(...);
    it = getFirstReachableConnection(it);
  }

  routes[i++] = new Route(legs);
}
```

Code 2.11: Journey Extraction door middel van post-processing

```
private StationQuintuple getFirstReachableConnection(StationQuintuple arrivalquint) {
  List<StationQuintuple> it_options =
  ↪  S.get(arrivalquint.arrivalConnection.arrivalStationUri);
  int i = it_options.size() - 1;
    while (i >= 0 && it_options.get(i).arrivalTime.getMillis() !=
    ↪  arrivalquint.arrivalTime.getMillis() - 240 * 1000) {
    i--;
  }
  return it_options.get(i);
}
```

Code 2.12: Vinden van volgende vertrek bij tussenstop

2.3.2 Vertrekken en aankomsten per station

In tegenstelling tot routes, kunnen we om de zogenoemde *liveboards* te berekenen een eenvoudig algoritme gebruiken. We overlopen alle pagina's, en houden enkel de stops bij die betrekking hebben op het gezochte station. Hiervoor gebruiken we twee lijsten voor respectievelijk de vertrekkende en aankomende connecties. We blijven pagina's overlopen tot de stopvoorwaarde is bereikt. Deze verschilt afhankelijk van de locatie waar we het algoritme implementeren:

- Op een webserver blijven we extra pagina's van de schijf laden tot het gewenst aantal resultaten is bereikt, of het gewenste tijdsinterval overlopen is.
- Bij een lokale implementatie stoppen we zodra we de pagina hebben afgewerkt waarin de eerste stop beschreven wordt. We maken gebruik van incrementele resultaten, waarbij telkens zo snel mogelijk een klein resultaat wordt berekend. Hierdoor krijgt de gebruiker (in theorie) sneller resultaten te zien.

Wanneer de stopvoorwaarde is bereikt, splitsen we de lijsten in drie soorten stops:

- Vertrekhalte: dit zijn de stops van voertuigen die deze stopplaats als begin van hun traject hebben en dus geen informatie over een aankomst bevatten
- Eindhalte: dit zijn de stops van voertuigen die deze stopplaats als laatste op hun traject hebben en dus niet meer vertrekken
- Stops: dit zijn de stops van voertuigen die een tussenstop maken

Om te bepalen tot welke categorie een connectie behoort, dienen we voor elke aankomende connectie in het station te bepalen of er een vertrekkende connectie mee overeenkomt. Komt er een vertrekkende connectie mee overeen, is er sprake van een tussenstop. Komt er geen vertrek mee overeen, is dit een eindhalte. De connecties die vertrekken vanuit het station, en waarmee geen aankomst overeenkomt, zijn dan de vertrekhaltes.

Bij de implementatie hiervan zorgen we ervoor dat we voor elk vertrek alle aankomsten volgens dalende aankomsttijd overlopen: we verwachten in de meeste gevallen een tussenstop (en dus geen vertrek), en in het geval van een tussenstop zal de aankomst slechts enkele minuten eerder zijn. Door de aankomsten volgens dalende aankomsttijd te overlopen zullen we dus sneller de bijhorende aankomende connectie ontdekken, indien deze bestaat.

Het is duidelijk dat dit algoritme slechts éénmaal de lijst met connecties overloopt. Hieruit volgt dat de efficiëntie $O(n)$ is, met n het aantal overlopen connecties. Gezien we alle connecties voor een bepaald tijdsinterval overlopen, is n evenredig met de drukte van het vervoersnetwerk.

2.3.3 Route van een voertuig

Ook de route van een voertuig afleiden uit Linked Connections data is eenvoudig. Zowel de stops van een trein, als de connecties, zijn chronologisch geordend. We overlopen dus alle connecties, waarbij we telkens de laatst gebruikte connectie p en de huidige connectie c in variabelen bijhouden.

- Wanneer we de eerste connectie ontdekken die betrekking heeft op deze trein, is dit de vertrekhalte. We verwerken deze data en voegen dit toe aan de lijst met stops. We kopiëren c naar p.
- Wanneer p een connectie bevat, en c de hierop volgende connectie bevat, beschikken we over alle informatie om de tussenstop tussen p en c toe te voegen aan de lijst met stops.
- Wanneer alle connecties overlopen zijn, was c de laatste connectie van dit trein. Deze connectie duidt dus de eindhalte aan. We voegen deze toe aan de lijst met stops, en geven een antwoord terug aan de oproepende code.

Het bepalen van de route van een trein legt een eerste mogelijk pijnpunt van de Linked Connections-specificaties bloot: hiervoor moeten alle connecties voor een volledige dag opgehaald worden. Er is immers geen enkele manier om te bepalen wanneer een trein vertrekt, of wat de eindhalte is. Hierdoor is er een relatief lange laadtijd om alle fragmenten op te halen, wat enigszins beperkt kan worden door deze asynchroon te laden. Het effect hiervan op de gebruikerservaring zullen we later onderzoeken.

Het is duidelijk dat dit algoritme slechts éénmaal de lijst met connecties overloopt. Hieruit volgt dat de efficiëntie $O(n)$ is, met n het aantal overlopen connecties. Gezien we net zoals bij liveboards alle connecties voor een bepaald tijdsinterval overlopen, is n ook hier evenredig met de drukte van het vervoersnetwerk.

2.4 Implementatie in HyperRail

De in sectie 2.3 besproken algoritmes zijn zowel server-side als client-side in de HyperRail applicatie geïmplementeerd. Door het gebruik van dezelfde algoritmes sluiten we uit dat een verschil in dataverwerking een verschil in gebruikerservaring kan veroorzaken. Om ook beïnvloeding door hardware uit te sluiten, wordt LC2Irail uitgevoerd op dezelfde server als graph.irail.be, waardoor beide processen dezelfde bronnen en belasting delen.

Om beide varianten te implementeren in de Android applicatie wordt voor elke variant een klasse

aangemaakt die de interface *IrailDataProvider*[9] implementeert. Deze interface stelt de nodige data ter beschikking van de applicatie. De volgende data worden vereist van klassen die deze interface implementeren:

- Vertrekken en aankomsten van voertuigen in een stopplaats
- Routes tussen twee stopplaatsen
- De voorgaande of volgende resultaten voor een resultaat
- Het traject van een voertuig
- De huidige storingen op het netwerk

Gezien de huidige storingen op het vervoersnet (nog) niet beschikbaar zijn in Linked Connections, zullen we voor deze functionaliteit telkens terugvallen op de reeds bestaande implementaties op basis van api.irail.be[10].

Het laden van voorgaande of volgende resultaten is een belangrijke methode. De applicatie zal alle resultaten voorstellen in een lijst die verlengd wordt zodra de gebruiker het einde ervan bereikt, zogenaamd *infinite scrolling*. Deze functie maakt ook de implementatie van incrementele resultaten op een eenvoudige wijze mogelijk. Hiervoor zal de implementatie die de algoritmes lokaal toepast telkens stoppen bij de pagina waarin zich het eerste resultaat bevindt. Een mogelijke manier om dit verder te versnellen bestaat eruit om telkens een aantal pagina's te *prefetchen*, zodat deze reeds gecachet zijn voor de verzoeken die er kort op volgen. Dit wordt dan ook toegepast voor de lokale Linked Connections implementatie: bij het opstarten wordt onmiddellijk data voor de komende 90 minuten geladen.

2.4.1 Bepalen van de parameters voor server-side implementatie

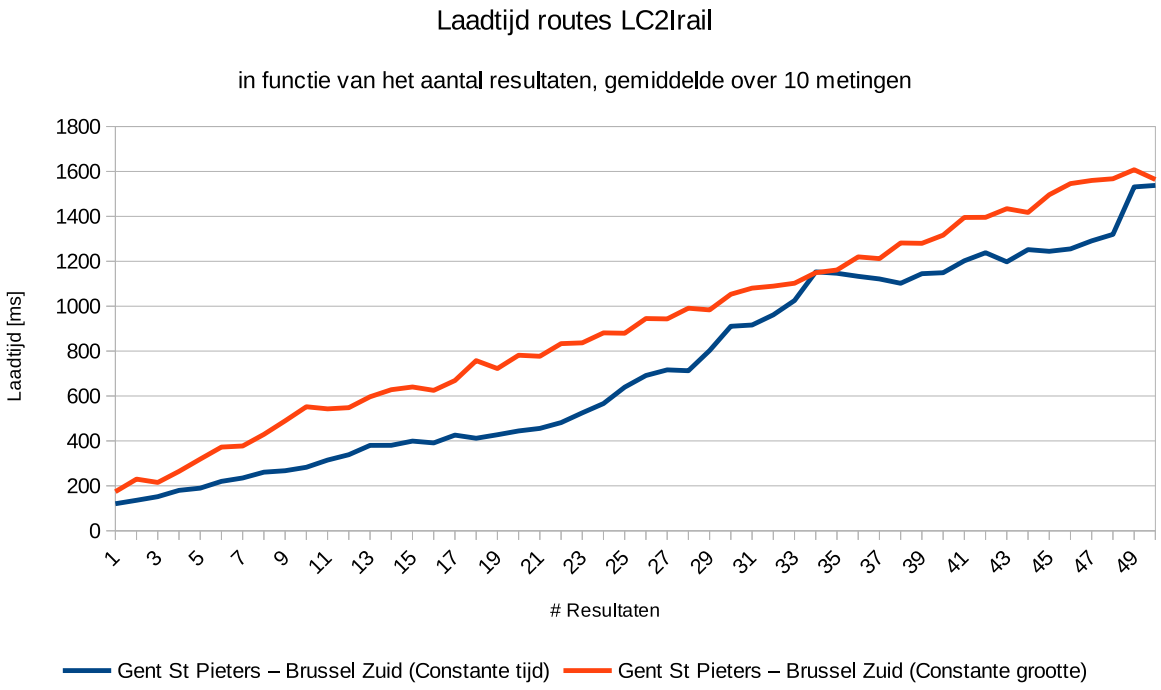
De implementatie waarbij algoritmes op de server toegepast worden, zal steeds een bepaald aantal resultaten proberen laden, om zo de overhead die HTTP requests met zich meebrengen te beperken. Hierbij moet er getracht worden om een evenwicht te vinden tussen een lange wachttijd, en het te vaak moeten opvragen van volgende resultaten, wat ook telkens een overhead met zich meebrengt.

Tijdens het schrijven van deze masterproef werd ook de Linked Connections server voor de NMBS nog verder ontwikkeld. Zo werd er afgestapt van het principe om pagina's van constante

[9] `https://github.com/Bertware/linkedconnections-android-client/blob/master/Hyperrail/src/main/java/be/hyperrail/android/irail/contracts/IrailDataProvider.java`

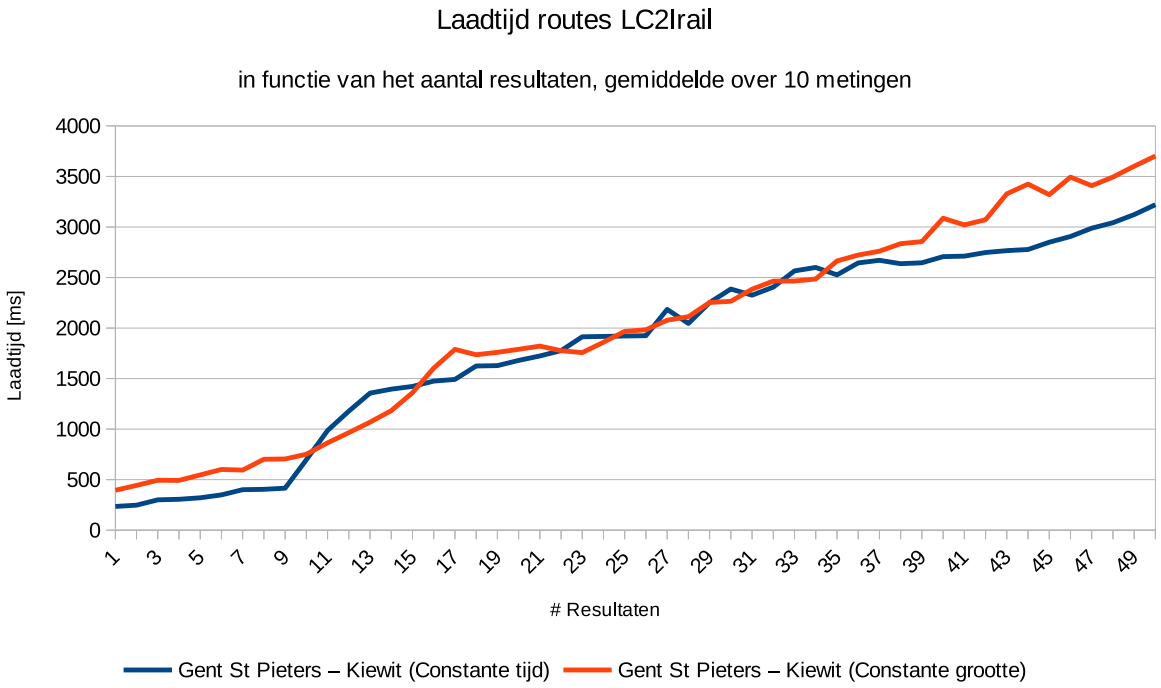
[10] `https://docs.irail.be/`

tijdsintervallen te gebruiken, en overgestapt naar pagina's van constante grootte. Pagina's die constante tijdsintervallen beschrijven hebben een aantal nadelen, zoals het feit dat pagina's 's nachts bijna leeg zijn (of de pagina's zelfs niet bestaan). De ongelijke grootte van de pagina's zorgt er ook voor dat de tijd nodig om een interval te doorlopen niet lineair is. Wanneer pagina's van constante grootte gebruikt worden, vallen deze nadelen weg: elke pagina bevat ongeveer evenveel resultaten. Het verschil tussen deze twee varianten is duidelijk zichtbaar in grafieken 2.1 en 2.2. Naast de laadtijd bij het gebruik van pagina's van constante grootte en constant tijdsinterval, is hier ook de laadtijd bij pagina's van constante grootte zichtbaar wanneer deze op de server in het geheugen gecachet worden. We zien hierbij een drastische verlaging van de responstijd, met iets meer fluctuaties door cache hits en misses. Hierbij blijkt de server wel extreem goed te presteren.



Figuur 2.1: De tijd die nodig is om routes te laden, met aankomst om 18u, in functie van het gewenste aantal resultaten.

We zien dat deze tijd telkens ongeveer lineair toeneemt, maar zeker de variant op basis van constante tijdsintervallen sterke knikken vertoont op sommige plaatsen. Deze knikken kunnen we verklaren aan het overlopen van pagina's met weinig connecties, zoals 's nachts wanneer geen treinen rijden en tijdens daluren. Sommige knikken die we in beide grafieken zien, worden veroorzaakt door pagina's met weinig relevante connecties. Dit is duidelijk merkbaar wanneer beide grafieken vergeleken worden: Tussen Gent en Brussel Zuid (een belangrijke treinverbinding) zijn er geen knikken wanneer gebruik gemaakt wordt van pagina's met een constante grootte, terwijl deze er wel zijn wanneer een route naar een klein station zoals Kiewit gepland wordt.

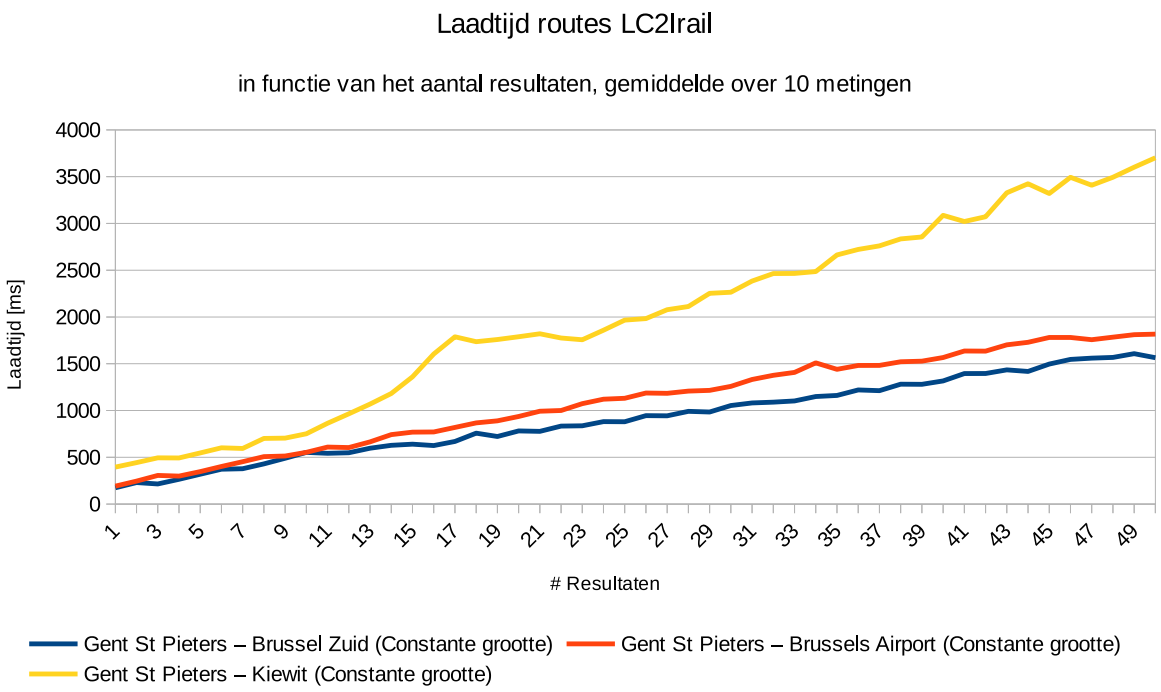


Figuur 2.2: De tijd die nodig is om routes te laden, met aankomst om 18u, in functie van het gewenste aantal resultaten.

Deze verschillen zijn zichtbaar in grafiek 2.3. Om te voorkomen dat gebruikers te lang moeten wachten op een resultaat wanneer een route naar een klein station gepland wordt, zal de server implementatie steeds 8 resultaten weergeven.

Liveboards, die informatie geven over vertrekkende en aankomende treinen in een station, bouwen we op voor een bepaald interval. Gebruikers willen steeds informatie over treinen in de komende periode, en niet enkel de eerste resultaten, gezien ze niet op voorhand weten de hoeveelste trein ze zoeken. Ook voor liveboards onderzoeken we nu hoe lang het duurt om een deze op te bouwen voor een gegeven tijdsinterval.

In grafiek 2.4 zien we de tijd die benodigd is om een liveboard te genereren over een bepaald interval. In deze grafieken zien we dat bij pagina's van constante tijdintervallen de responstijd initieel sterk stijgt, waarna de curve een lineaire vorm aanneemt. Knikken in de curve kunnen we opnieuw wijten aan daluren, in dit geval zien we een vlak segment tussen 800 en 960 minuten. Dit komt ongeveer overeen met de periode tussen 1 en 4 uur 's nachts, waarin zeer weinig treinen rijden. Wanneer we in detail gaan kijken naar het aantal stops per interval van 30 minuten, zichtbaar in grafiek 2.5, zien we duidelijk dat er een aantal uitschieters zijn rond de piekuren en 's middags, en momenten waarop er aanzienlijk minder treinen vertrekken, zoals in daluren en 's nachts. Deze fluctuaties zullen een effect hebben op zowel de server-side als

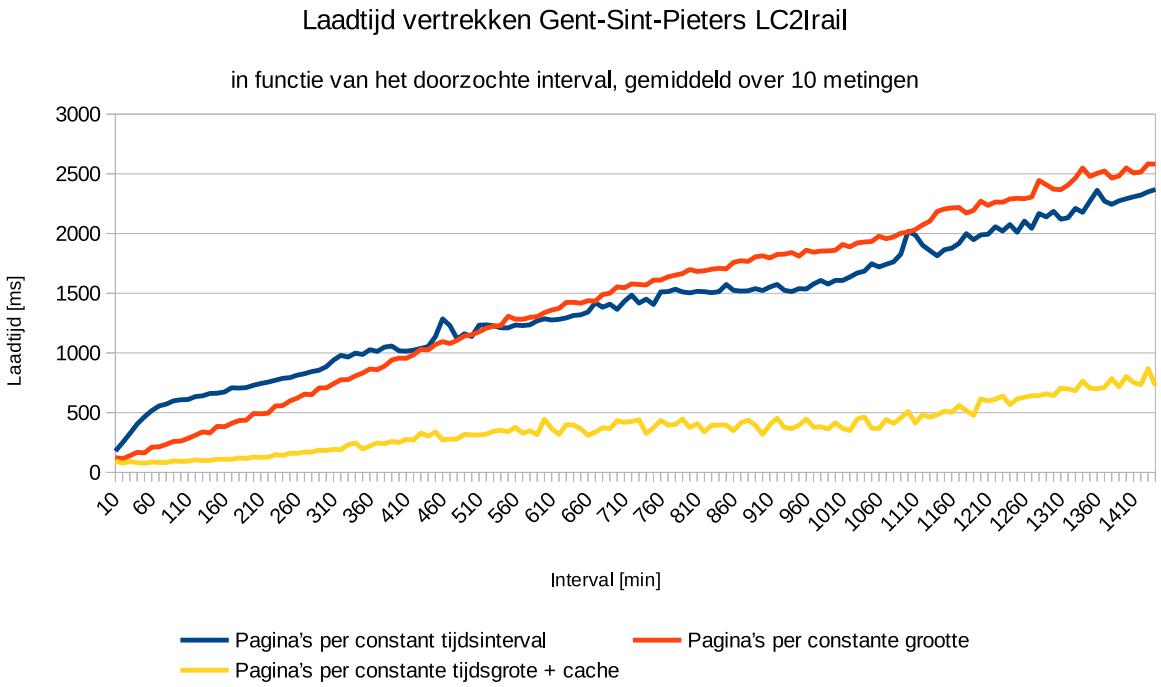


Figuur 2.3: De tijd die nodig is om routes te laden, met aankomst om 18u, in functie van het gewenste aantal resultaten. We zien een knik wanneer Linked Connections voor de nacht opgehaald worden, terwijl deze relatief weinig data bevatten.

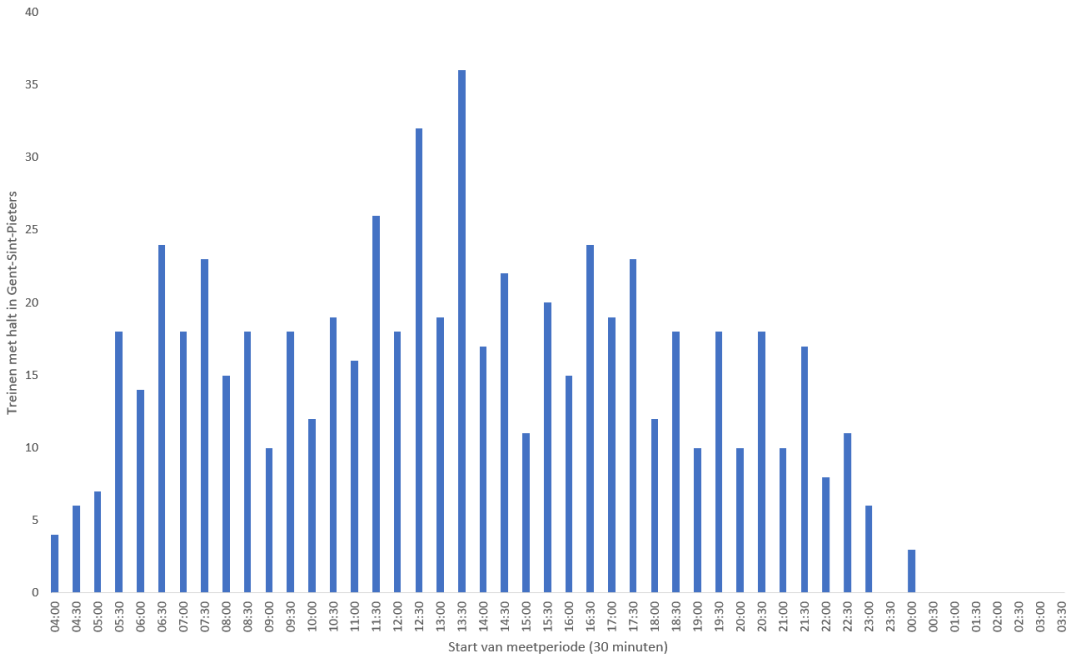
client-side implementaties. Mogelijk zou dit ervoor kunnen zorgen dat de gebruikerservaring voor zoekopdrachten in de spits beter is dan voor zoekopdrachten 's nachts.

Wanneer we echter kijken naar de laadtijd bij pagina's van constante grootte, verloopt deze curve bijna lineair. De curve bevat geen vlakke segmenten meer, gezien alle pagina's aanwezig zijn. De toenemende laadtijd op momenten dat er geen treinen rijden is te wijten aan de implementatie: elke geladen pagina wordt volledig verwerkt. Bij gebruik van pagina's met een constante grootte zullen deze implementaties 's nachts dus resultaten geven die verder in de toekomst liggen, in plaats van geen resultaten. Wanneer gebruik gemaakt wordt van caching zien we dat de responstijd verder drastisch verlaagt. Het is deze variant die in productie gebruikt zal worden.

Het is belangrijk om op te merken dat de metingen voor bovenstaande grafieken steeds gemaakt zijn op een lokale server, gebruik makend van krachtige hardware. Op zwakkere hardware zal de responstijd lineair hoger liggen. De LC2Irail applicatie wordt ingesteld om telkens 8 routes, en 60 minuten van vertrekken en aankomsten weer te geven. Hierdoor krijgt de gebruiker telkens genoeg resultaten, en wordt onnodig werk en tijdsverspilling vermeden.



Figuur 2.4: De tijd die nodig is om liveboards te laden over een bepaald interval.



Figuur 2.5: Het aantal voertuigen die stoppen in Gent-Sint-Pieters op een werkdag.

2.4.2 Caching

Een van de grootste voordelen die bij Linked Connections hoort, is de hoge mate waarin data gecachet en hergebruikt kan worden. Om data te cachen zullen we een SQLite database bijhouden op de interne opslag van het toestel.

Door gebruik te maken van een opslag op het toestel blijven alle opgeslagen pagina's behouden wanneer de applicatie afgesloten en herstart wordt. Hierdoor is de applicatie ook na het afsluiten nog te gebruiken voor offline opzoekingen. Wanneer pagina's constante tijdsintervallen beschrijven is het eenvoudig om te controleren of een pagina reeds gecachet is: de gezochte tijd afronden, URL construeren en opzoeken in de database is voldoende. Wanneer echter pagina's van constante fysieke grootte gebruikt worden, kan het id van de pagina niet meer client-side bepaald worden. Wanneer een nog nooit eerder opgezocht tijdstip gezocht wordt, zoals 8:05, is het mogelijk dat deze data zich in de pagina beginnend op 8:03 bevindt. Om dit op te lossen gebruiken we een query, waarbij de URL voor een tijdstip geconstrueerd wordt en deze vergeleken wordt met de intervallen per pagina. De keuze voor SQLite laat toe om dit eenvoudig te implementeren. Een voorbeeld hiervan is te zien in fragment 2.13. In de cache slaan we steeds de URL, het volledige server antwoord en de tijd waarop het antwoord ontvangen werd op.

```
db.query(TABLE, new String[]{"url", "data", "datetime"}, "url<=? AND next>?", new
↪  String[]{url,url}, null, null, "url DESC");
```

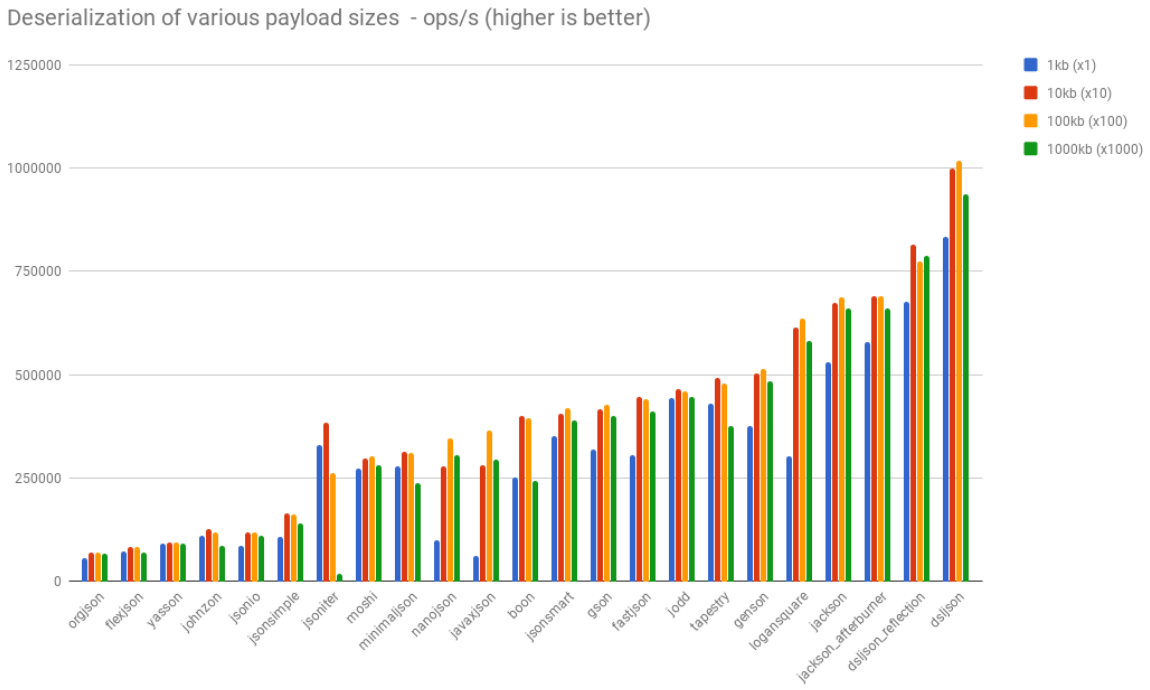
Code 2.13: SQLite query om juiste pagina in cache te zoeken

Deze cache wordt gecontroleerd alvorens een verzoek te maken. Indien geen internet beschikbaar is, of de gecachete data minder dan 60 seconden oud is, wordt deze hergebruikt. Deze maximale ouderdom kan ook uit de HTTP-headers van de antwoorden onttrokken worden. De HTTP antwoord headers zijn echter niet onmiddellijk toegankelijk in de gebruikte bibliotheek, en om de ontwikkeltijd te verkorten, werd deze in het kader van deze masterproef echter hardgecodeerd.

2.5 Optimalisatie verwerking op een mobiel toestel

Bij de implementatie van Linked Connections in Android is vooral aandacht besteed aan de algoritmes. Efficiënte algoritmes zijn immers belangrijk voor het snel beantwoorden van de zoekopdracht door de gebruiker. Linked Connections parsen van JSON naar objecten wordt gedaan aan de hand van de standaard JSON parser in Android, org.json. Alle connecties van een pagina worden volledig geparset, alvorens de pagina verwerkt wordt door algoritmes. Tijdens user testing wordt het echter al snel duidelijk dat – hoewel de app goed presteert op enkele testtoestellen en een emulator – er ook verschillende gebruikers zijn bij wie de app enorm slecht presteert. Analyse

van het CPU-gebruik door de applicatie[11] duidt al snel aan dat er veel tijd besteed wordt aan het parsen van JSON, en ook specifiek aan het parsen van datums. Hierdoor duren opzoekingen, zowel online als offline, langer dan nodig. Het effect van de trage JSON parser wordt uitvergroot op tragere toestellen, waarbij het opzoeken van voertuigen enkele seconden langer kan duren vergeleken met een sneller toestel, ook als beide toestellen alle gegevens reeds in cache hebben. Als oplossing wordt in eerste instantie het parsen van datums versneld: het tijdsformaat wordt expliciet gedefinieerd in een statische variabele, zodanig dat parsen hier optimaal verloopt. Ten tweede worden niet meer alle connecties volledig geparset: de ongeparsete JSON wordt opgeslagen in het LC-object, en velden worden slechts geladen wanneer nodig. Dit wil zeggen dat in het geval van liveboards enkel de vertrektijd en het vertrek- en eindstation van elke connectie geladen worden, in plaats van alle velden. Pas wanneer een connectie relevant wordt en andere velden nodig zijn, worden deze uit de JSON data geladen. Deze aanpassingen zorgen voor een kleine verbetering, maar hebben geen al te grote impact: het laden van een voertuig gaat van gemiddeld 4500ms naar 3900ms op een HTC 10. Dit blijft enorm veel vanuit het standpunt van een gebruiker. Wanneer we echter kijken naar een vergelijking tussen JSON parsers, blijkt al snel dat de standaard *org.json* parser bij de traagste parsers voor Java hoort[12]. Dit is duidelijk te zien in figuur 2.6.



Figuur 2.6: Prestaties van verschillende JSON parsers bij deserialiseren. ©2016 Fabien Renaud

[11] CPU verbruik kan niet exact gemeten worden, maar een indicatie van welke methodes het meest CPU tijd vereisen zijn wel mogelijk. Dit wordt verder besproken bij de beperkingen van dit onderzoek, sectie 4.8

[12] `https://github.com/fabienrenaud/java-json-benchmark`

Als oplossing is gekozen om de *LoganSquare* parser te gebruiken. *LoganSquare* is niet de snelste, maar wel aanzienlijk sneller dan *org.json* en relatief eenvoudig te implementeren. Deze parser is gebaseerd op Jackson, waardoor het grootste voordeel van Jackson gedeeld wordt: achterliggend wordt gebruik gemaakt van streaming, waardoor minder *garbage* veroorzaakt wordt. Hierdoor is minder *garbage collection* nodig, waardoor de code minder vaak gepauzeerd hoeft te worden en dus sneller resultaten geeft. De verschillen in performantie tussen de parsers worden besproken in hoofdstuk 4. Binnen de beperkte tijd die beschikbaar was voor het schrijven van deze masterproef was er onvoldoende tijd om *DSLJson* te implementeren. Afgaand op figuur 2.6 zou hier echter wel nog prestatiewinst geboekt kunnen worden.

"Alice: How long is forever? White Rabbit: Sometimes, just one second. — Alice's Adventures in Wonderland"

~Lewis Carroll

3

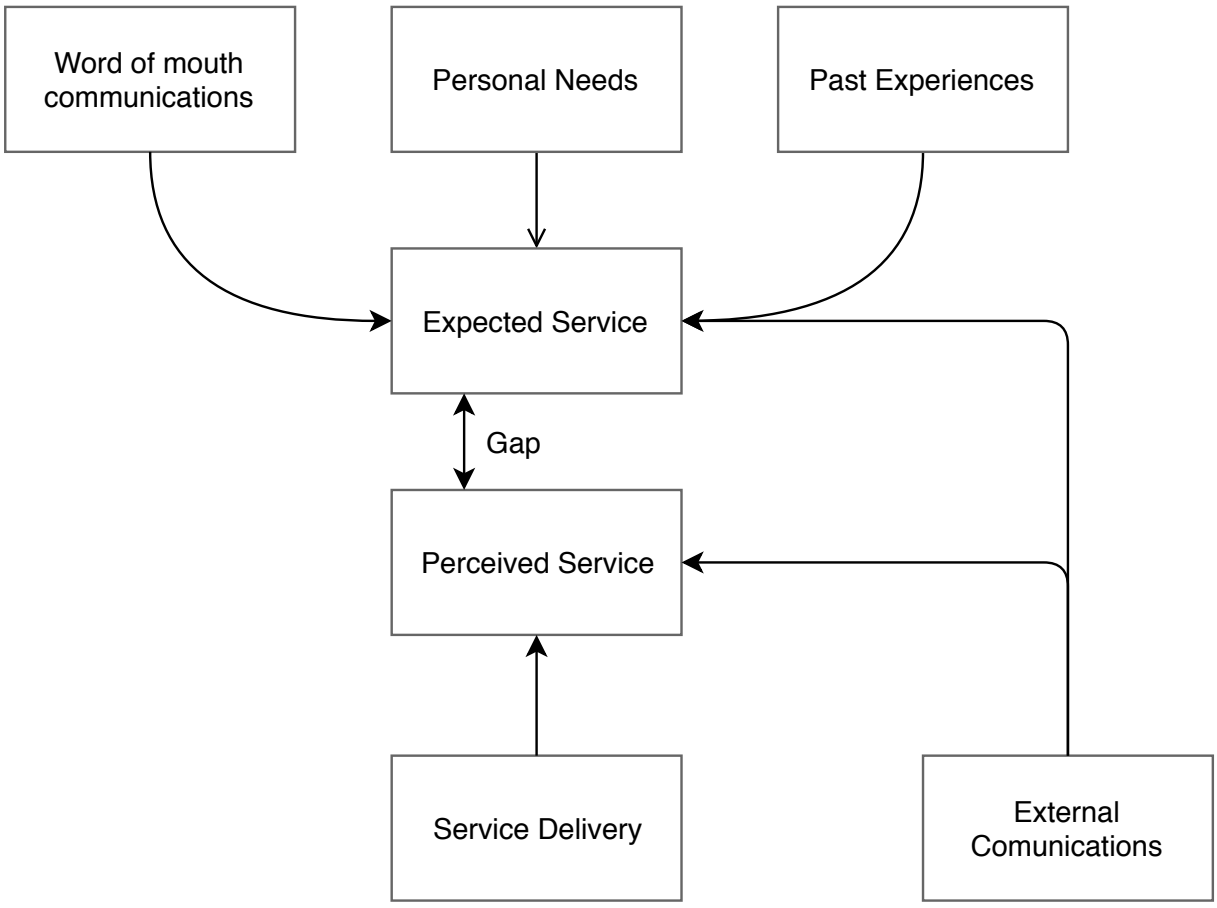
Evaluatie

De user-perceived performantie kan afwijken van de werkelijke performantie. Dit verschil wordt voornamelijk bepaald door de interface van de applicatie. Zo ervaren gebruikers wachttijden als minder lang wanneer ze een gevoel van vooruitgang krijgen, door het gebruik van progress bars of andere visuele hints. Aangezien we bij dit onderzoek dezelfde applicatie voor beide API's gebruiken, zullen deze visuele hints de vergelijking van de user perceived performance tussen de twee diensten niet beïnvloeden. Echter biedt Linked Connections de mogelijkheid tot incrementele resultaten: nog tijdens het laden kunnen al snel eerste resultaten weergegeven worden, waardoor de gebruiker deze techniek als sneller kan ervaren, ook al duurt het langer om een gelijk aantal resultaten op te halen. Ook voor LC2Irail zal de applicatie van incrementele resultaten gebruik maken, echter is het hier steeds de server die bepaalt wanneer data verzonden worden. De server zal steeds pogen om een bepaald tijdsinterval te verwerken, of een bepaald aantal resultaten te vinden alvorens de client te antwoorden. Hierdoor kunnen de resultaten wel per tijdsinterval getoond worden, maar niet zo atomair als wanneer Linked Connections gebruikt wordt.

In figuur 3.1 zien we de factoren die bijdragen tot de verwachting van de gebruiker en de ervaring van de gebruiker. De User Experience wordt hier tot de *service delivery* gerekend. Naast het verschil tussen de werkelijke geleverde service en de ervaren service, merken we ook een verschil tussen de verwachte service en de ervaren service, in Engelstalige literatuur omschreven als een 'service quality gap'. De verwachtte service is wat de gebruiker verwacht van de applicatie, onder

andere in termen van snelheid, mogelijkheden en dataverbruik. Dit correleert duidelijk met de User Experience: eerder onderzoek wijst uit dat interface design, prestaties van de applicatie, batterijgebruik, kostprijs van de applicatie en connectiviteit, gebruiksersroutines en levensstijl invloed hebben op de user experience [11]. In dit onderzoek zullen we ons specifiek richten op de prestaties, batterijverbruik, dataverbruik en offline beschikbaarheid.

De gebruiker heeft geen exact beeld over batterijverbruik en dataverbruik, maar wel over snelheid en offline beschikbaarheid. Of de gebruiker Linked Connections als snel ervaart, zal indirect bepaald worden door de snelheid van alle andere applicaties die hij gebruikt (past experiences) en de tijd die hij kan wachten op een antwoord (personal needs). Zo vinden we het bijvoorbeeld normaal dat een video uploaden enkele minuten kan duren, terwijl we verwachten dat het verzenden van een chatbericht dat enkel uit tekst bestaat onmiddellijk gaat. In dit geval is de verwachtte service gebaseerd op de bestaande applicaties, welke allemaal ongeveer gelijke prestaties bieden en gebruikmaken van RPC API's. De personal need is om onmiddellijk informatie op te zoeken, zonder veel vertraging. Met andere woorden verwacht de gebruiker om binnen enkele seconden de informatie te krijgen.



Figuur 3.1: Factoren die bijdragen tot de verwachtingen van de gebruiker

Tijdens dit onderzoek zullen we zowel de *service delivery* als *perceived service* proberen vergelijken tussen de twee technieken. Door deze vergelijking zullen we kunnen vaststellen of Linked Connections een betere user-perceived performance biedt, zowel absoluut als relatief ten opzichte van de werkelijke performance.

Een diepgaand onderzoek over de verwachte service valt buiten het bereik van deze masterproef. We zullen echter wel bondig de verwachtingen van gebruikers ondervragen, op vlak van dataverbruik, offline functionaliteit, en privacy, gezien deze drastisch verschillen bij Linked Connections ten opzichte van meer traditionele API's.

3.1 Objectieve metingen

Met objectieve metingen zullen we de werkelijke performance van elke API vastleggen. Ook zullen we het dataverbruik en batterijverbruik van beide technieken aan de hand van deze metingen trachten te bepalen.

Aangezien het onmogelijk is om automatisch volledige zoekopdrachten uit te voeren door de applicatie zonder de benodigde tijd te beïnvloeden, en aangezien de benodigde tijd om te renderen een constante is die gelijk is voor alle implementaties, zal er rechtstreeks op de API implementatie getest worden, net zoals de API normaal gezien gebruikt wordt. Het uittekenen van de resultaten op het scherm is een constante, welke verwaarloosbaar klein is in vergelijking met de tijd benodigd voor het ophalen van resultaten.

Om te voorkomen dat de keuze van de geteste stations of routes de objectieve metingen vertekent, zullen we de opzoekingen van echte gebruikers gebruiken. Hiervoor gebruiken we de log data van api.irail.be, die publiek beschikbaar zijn[1]. Door deze queries opnieuw af te spelen op de applicaties kunnen we een zo goed mogelijk beeld krijgen van de werkelijke prestaties.

Objectief zullen we proberen om volgende gegevens vast te leggen

- de gemiddelde tijd om alle data van de server te halen
- de gemiddelde tijd tussen zoekopdracht en weergave van het eerste resultaat
- de gemiddelde tijd tussen zoekopdracht en weergave van de eerste tien resultaten
- de gemiddelde hoeveelheid data die verzonden en ontvangen wordt
- het gemiddelde processorgebruik van het toestel
- het gemiddelde batterijgebruik van het toestel

[1] https://gtfs.irail.be/logs

Hiertoe zullen we gebruik maken van een HTC 10 en een HTC One. Dit zijn twee smartphones in een zeer verschillende klasse op vlak van hardware én software. De verschillen tussen beide toestellen worden verduidelijkt in tabel 3.1

Kenmerk	HTC One (M7)	HTC 10
Besturingssysteem	Android 5.0	Android 8.0
Processor	Qualcomm Snapdragon 600 (4x Krait 1,7GHz)	Qualcomm Snapdragon 820 (4x Kryo 2,2GHz)
Werkgeheugen	2GB	4GB
Wi-Fi	802.11ac	802.11ac
Antutu Benchmark	31.447 (mediaan)	133.791 (top 10%)

Tabel 3.1: De specificiaties van gebruikte testtoestellen.

Benchmarks zijn synthetisch, en puur ter indicatie. Zo scoren typische budgettoestellen zoals de Motorola Moto G (1st gen), Motorola Moto G (3rd gen) en Nokia 3 respectievelijk 17.223, 21.000 en 26.500 op dezelfde AnTuTu benchmark, waarmee ze iets lager uitkomen dan de HTC One. Deze toestellen zijn instapmodellen uit 2013 tot nu. Een modern toestel van €200, zoals de Nokia 5, scoort 40.000, en komt daarmee al ruim boven de HTC One uit op vlak van prestaties. De HTC One is hierdoor geschikt om de prestaties te testen op oude en low-end smartphones.

Aan de andere kant is er de HTC 10, welke een gelijkaardige score behaalt als de Nokia 7 (103.000) en Samsung Galaxy S7 (157.000), en een (aanzienlijk) lagere score dan de Nokia 8 (208.000) of Samsung Galaxy S8 (201.000). Deze smartphone is dus geschikt om de prestaties om te testen voor duurdere mid-range toestellen of oudere high-end toestellen. Recentere high-end toestellen presteren aanzienlijk beter, oudere of goedkopere mid-range toestellen vallen tussen de HTC One en HTC 10 in.

Zoals eerder vermeld zullen de opzoekingen van echte gebruikers worden gebruikt om te voorkomen dat het onderzoek vertekend wordt. In dit geval werden de opzoekingen op 2 mei 2018 gebruikt.

Voor alle tests werd gebruik gemaakt van de HTC 10 of HTC One zoals besproken in hoofdstuk 3, verbonden met internet via wifi (ping 26ms, downloadsnelheid 42mbps, uploadsnelheid 9mbps). Het toestel werd niet gebruikt tijdens de testen, en er werden geen achtergrondapplicaties uitgevoerd. De metingen werden automatisch uitgevoerd met behulp van *instrumented tests*. Metingen van Linked Connections (LC) gebruiken de LoganSquare JSON parser tenzij anders vermeld.

Voor de drie types resultaten zullen we telkens de metingen uit benchmarks bespreken, en de resultaten van user-testing. Bij de metingen zullen we telkens kort het effect van implementa-

tiedetails bespreken, waarna we specifiek en gedetailleerd de prestaties van de huidige implementatie, op basis van de LoganSquare parser, bespreken. Uit deze metingen zullen we telkens trachten specifieke oorzaken van prestatieverschillen te achterhalen.

3.2 Subjectieve metingen

Gezien Linked Connections nog enkele belangrijke gegevens mist, zoals of een stop al dan niet afgeschaft is, en aan welk perron het voertuig zal aankomen of vertrekken, kunnen we dit systeem nog niet zelfstandig door gebruikers laten testen. Om rond deze beperking heen te werken zullen we in plaats hiervan begeleide user-tests uitvoeren met gebruikers, waarbij gebruikers gevraagd wordt om hun gebruikelijke opzoekingen te doen, per implementatie hun mening te geven, en vervolgens te bevragen welke variant hun voorkeur geniet, op vlak van snelheid, functionaliteit, en privacy. We zullen ook zeer eenvoudig aftasten welke functionaliteit gebruikers het meest interessant vinden, zodat verder onderzoek zich hierop kan richten.

Door middel van een bevraging zullen we trachten een antwoord te vinden op volgende vragen:

- Biedt offline informatie een meerwaarde voor gebruikers?
- Hecht de gebruiker belang aan privacy bij het gebruik van routeplanning apps? Zo ja, in welke mate?
- Heeft de gebruiker schrik om te veel mobiele data te verbruiken?
- Hecht de gebruiker belang aan dataverbruik bij het gebruik van routeplanning apps?
- Is de gebruiker tevreden met de snelheid van zijn huidige routeplanning app?
- Wat is voor een gebruiker belangrijk in routeplanning apps?
- Is de gebruiker geïnteresseerd in routeplanning op maat? Zo ja, welke aspecten spreken hem dan aan?
- Is de gebruiker geïnteresseerd in offline opzoekingen?
- Is de gebruiker geïnteresseerd in de mogelijke snelheid die Linked Connections biedt?
- Is de gebruiker geïnteresseerd in de volledige privacy die Linked Connections biedt?

Door middel van user-testing zullen we proberen om ook deze vragen te beantwoorden:

- Ervaart de gebruiker een app die lokaal Linked Connections gebruikt als sneller dan een app die gebruik maakt van een RPC API?

- Ervaart de gebruiker een app die lokaal Linked Connections gebruikt als sneller dan zijn huidige app?

Om een antwoord op bovenstaande vragen te vinden, werd een enquete opgebouwd. Deze exacte vraagsteling voor deze enquete is terug te vinden in bijlage A.

"The most efficient network request is one that doesn't use the network."

~Roy T. Fielding

4

Resultaten

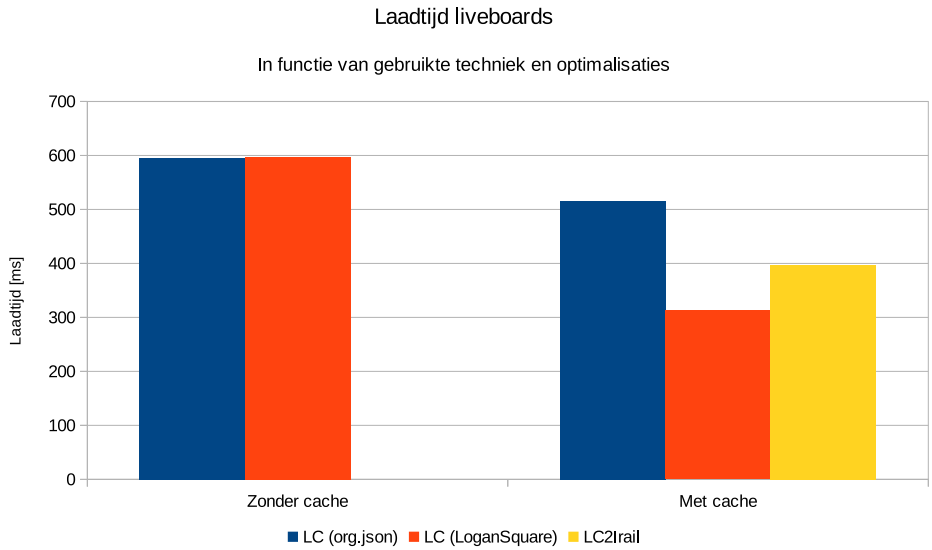
Hierna worden telkens de ervaringen van gebruikers besproken. Dit omvat absolute ervaringen (zonder exact referentiepunt, maar tegenover de expected service zoals vermeld in hoofdstuk 3), maar ook de relatieve ervaringen tussen LC2Irail en Linked Connections, en de relatieve ervaringen tegenover de huidige applicatie van de gebruiker zullen besproken worden.

Tot slot zullen we nog kijken naar de uiteindelijke keuze van de gebruiker, en bespreken we ook de resultaten van de enquête. Een globale interpretatie van de resultaten volgt in hoofdstuk 5.

4.1 Liveboards

4.1.1 Metingen

Zoals eerder vermeld vergelijken we eerst kort verschillende implementaties van Linked Connections. In grafiek 4.1 zijn de gemiddelde resultaten zichtbaar van een benchmark waarbij 262 stations opgezocht werden, ongeveer 5% van de opzoekingen door gebruikers op 2 mei 2018. Telkens is de minimale, gemiddelde en maximale responstijd gemeten, dit zowel gebruikmakend van de standaard (*org.json*) JSON parser en gebruikmakend van de *LoganSquare*-parser. Ook werd de test herhaald met cache in- en uitgeschakeld, om zo het effect hiervan te meten. Tot slot werd



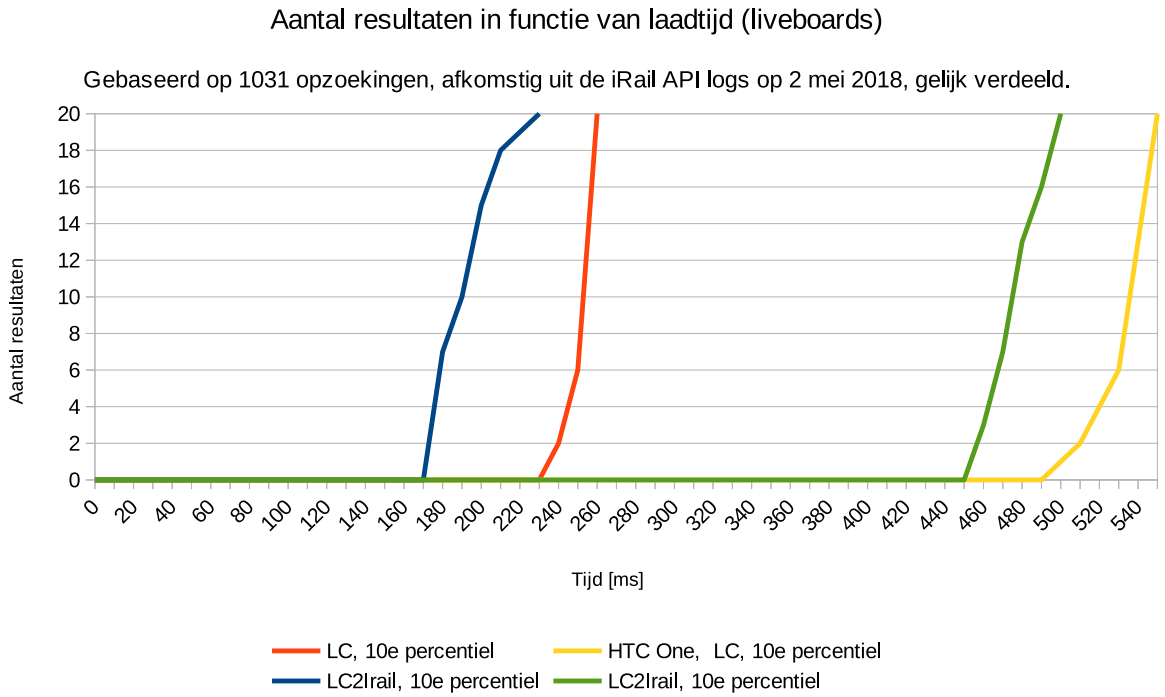
Figuur 4.1: De gemiddelde gemeten laadtijd voor liveboards gebruikmakend van een HTC 10 voor 262 opzoekingen gebaseerd op de iRail-logs.

dezelfde test herhaald gebruikmakend van data afkomstig van de LC2Irail web applicatie om een vergelijking tussen de twee methodes te kunnen maken. Deze cijfers geven slechts een indicatie van de snelheid – een volledige en diepgaande statistische analyse van de performantieverschillen tussen verschillende implementaties van dezelfde techniek valt wegens tijdsgebrek buiten het bereik van deze masterproef.

In deze cijfers is invloed van de cache duidelijk merkbaar. We zien wel een duidelijk verschil tussen de JSON parsers: terwijl bij gebruik van de *LoganSquare*-parser de gemiddelde laadtijd bijna halveert, is het effect van de cache bij het gebruik van de *org.json* parser veel kleiner. Wanneer de cache uitgeschakeld is, is het verschil tussen de parsers verwaarloosbaar. Dit is mogelijk te verklaren doordat voor het tonen van vertrekken of aankomsten er relatief weinig data nodig is: in de meeste gevallen volstaat een enkele Linked Connections-pagina.

Om een exact beeld te vormen van de prestaties, zoeken we een duizendtal liveboards op. Hiervoor kiezen we elke vijfde opzoeking uit de iRail-logs. Voor elk liveboard worden twintig resultaten geladen. De resultaten hiervan zijn zichtbaar in grafieken 4.2, 4.3 en 4.4, respectievelijk voor het tiende, vijftigste en negentigste percentiel. Uit deze grafieken kunnen we duidelijke trends zien:

- In de snelste gevallen is de serverimplementatie sneller. Hiervoor kunnen we verschillende oorzaken aanwijzen:
  - De serverimplementatie kan resultaten op een specifieke vraag cachen, terwijl de



Figuur 4.2: Het aantal resultaten in functie van de verlopen tijd.

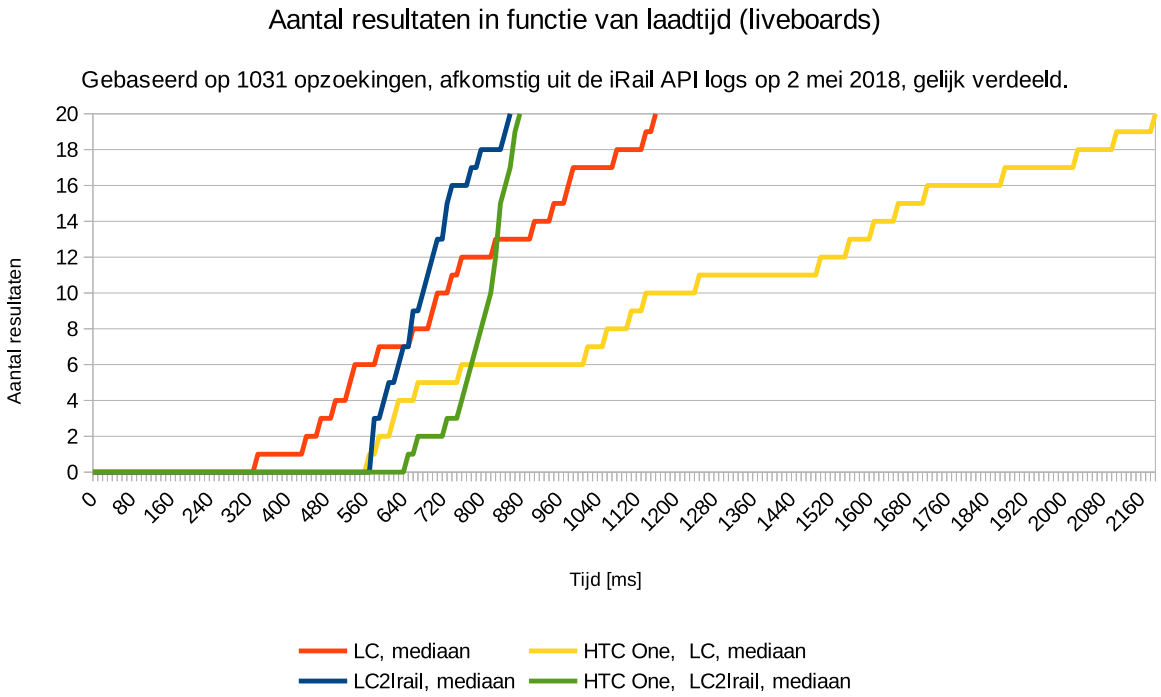
lokale implementatie deze steeds zal herberekenen vanaf Linked Connections pagina's. Voor veel voorkomende zoekopdrachten, zoals populaire stations, kan de server ook Linked Connections pagina's cachen.

– De serverimplementatie hoeft minder data te versturen. Ook het parsen van het antwoord gaat sneller, gezien slechts een kleine hoeveelheid data verwerkt moet worden en er verder geen berekeningen moeten gebeuren.

- Terwijl in de snelste gevallen de serverimplementatie sneller is, is dit verschil beperkt tot ongeveer 60 milliseconden voor het eerste resultaat. Dit ligt zeer kort bij een verschil van 10%, wat in deze ordegrootte moeilijk onderscheidbaar is voor gebruikers [25].

- Wanneer we naar de mediane performantie kijken, is Linked Connections duidelijk sneller voor de eerste resultaten. Dit snelheidsverschil is aanzienlijk, en hoogstwaarschijnlijk vooral te wijten aan het feit dat Linked Connections volledige ondersteuning biedt voor incrementele resultaten, waarbij de server meestal meerdere pagina's zal overlopen voor een antwoord gegeven wordt. Dit is te zien aan de steile curves voor LC2Irail, waar de curves voor LC gekenmerkt worden door een minder sterke stijging.

  Anderzijds is er bij Linked Connections sprake van een *overhead* door het laden van te veel data. Voor stations met weinig vertrekken, of tijdens piekuren, kan het hierdoor meer moeite kosten om Linked Connections te verwerken. We vermoeden dat dit contrast met

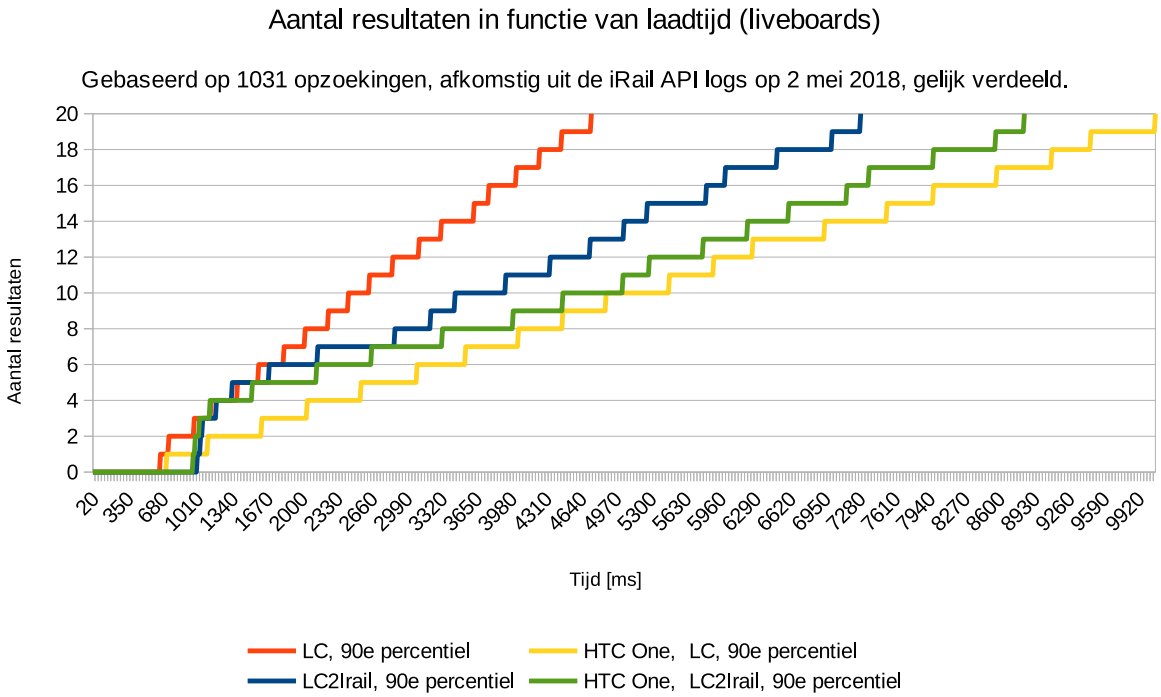


Figuur 4.3: Het aantal resultaten in functie van de verlopen tijd.

de korte, informatiedichte antwoorden van de RPC API zorgt voor het trager laden van resultaten. We zien dat hoe sneller het toestel is, hoe langer Linked Connections het snelst blijft. Dit bevestigt de hypothese dat het verwerken van Linked Connections pagina's aan de oorzaak ligt. Ook blijft het verschil tussen Linked Connections en LC2Irail beperkt op de HTC 10, terwijl dit verschil aanzienlijk oploopt op de tragere HTC One.

- In alle gevallen is er een sterke gelijkenis tussen de curves voor LC2Irail op het HTC 10 toestel en het HTC One toestel. Deze zijn enkel een relatief kleine afstand in tijd verschoven, wat verklaard kan worden door de lage belasting voor het mobiele toestel wanneer een RPC API gebruikt wordt.
- Het verschil in laadtijd tussen Linked Connections op de twee toestellen loopt lineair op met de benodigde laadtijd om resultaten te laden. Zo zien we dat in het slechtste geval dubbel zoveel tijd benodigd is op de HTC One in vergelijking met de HTC 10. Dit is een sterke indicatie dat de performantie van Linked Connections afhankelijk is van het gebruikte toestel.

Wanneer we naar de spreiding van de laadtijden kijken, zichtbaar in figuur 4.5 voor het eerste resultaat en figuur 4.6 voor het tiende resultaat, zien we ook hier duidelijke verschillen tussen de verschillende methodes. Telkens levert LC2Irail een dichtere verdeling op dan LC, waarbij Linked Connections enkel sneller is voor het eerste resultaat op de HTC 10. Voor het eerste

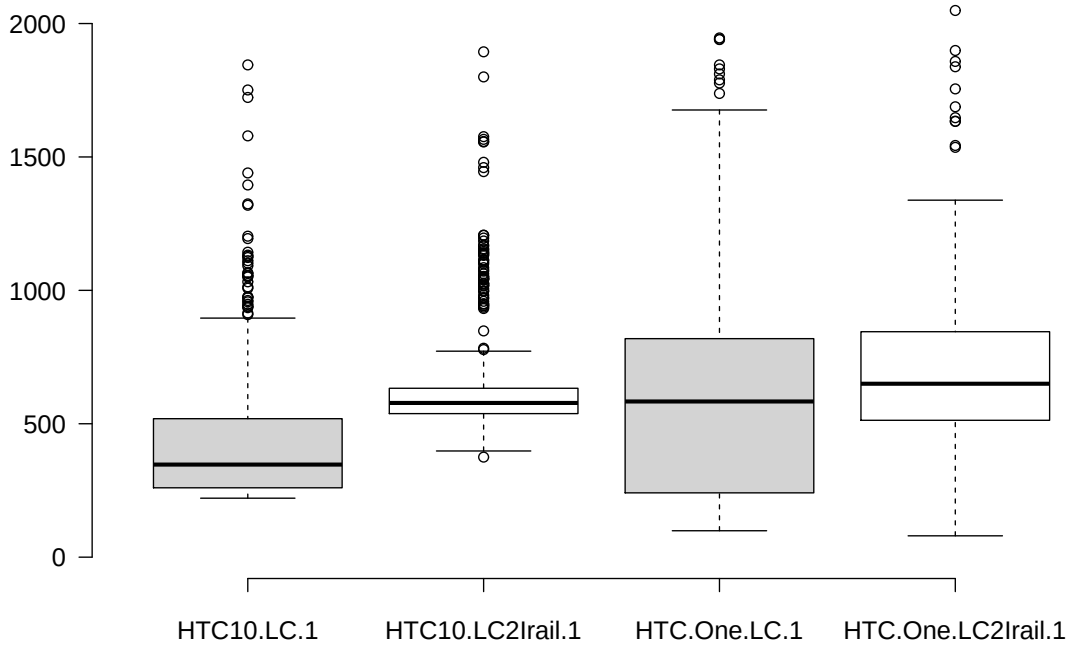


Figuur 4.4: Het aantal resultaten in functie van de verlopen tijd.

resultaat op de HTC One, en het tiende resultaat op de HTC 10, zijn beide implementaties volgens de wet van Weber-Fechner niet te onderscheiden op de mediaan.

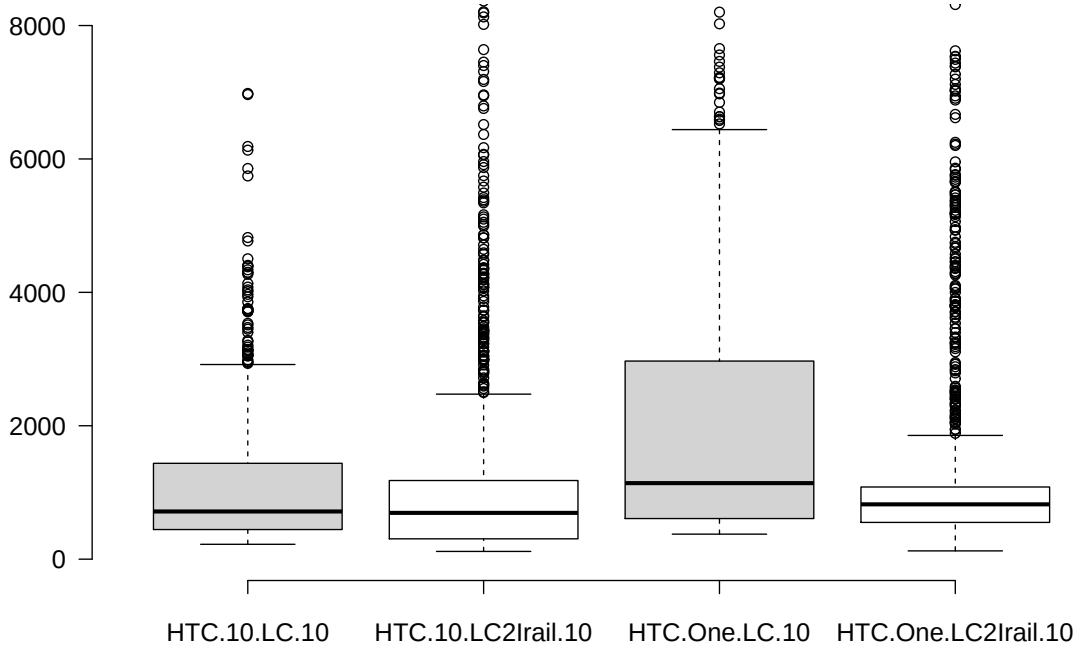
In deze grafieken zijn ook de verschillen tussen toestellen enorm interessant. Zo zien we dat bij gebruik van een RPC API de mediaan van de laadtijd ongeveer gelijk is tussen verschillende toestellen, en de interkwartielafstand relatief klein is, wat op een kleine spreiding en dus consistente resultaten per toestel duidt. De mediaan voor beide toestellen ligt respectievelijk op 578 en 650 milliseconden voor het eerste resultaat. Volgens de wet van Weber-Fechner is dit onmerkbaar voor gebruikers. Ook het eerste kwartiel ligt kort genoeg bij elkaar om niet merkbaar te zijn. Ondanks dat we boven de mediaan zien dat opzoekingen via LC2Irail op de HTC One meer tijd vergen, is onder de mediaan voor gebruikers geen verschil merkbaar. Deze zeer consistente ervaring over toestellen is zeer wenselijk voor routeplanning applicatie. Voor het tiende resultaat neemt het verschil tussen de toestellen toe, waarbij gebruikers een verschil zullen merken tussen de LC2Irail implementatie op verschillende toestellen.

Bij gebruik van Linked Connections zien we duidelijke verschillen tussen de mediaan van de laadtijd bij verschillende toestellen. Ook de interkwartielafstand varieert tussen toestellen: zo is een ouder toestel niet enkel trager, maar is ook de spreiding veel groter, en zijn de resultaten dus minder consistent op oudere (tragere) toestellen. Anderzijds is Linked Connections in de meeste gevallen wel duidelijk sneller dan LC2Irail op de HTC 10.



Figuur 4.5: Laadtijd eerste resultaat liveboard in functie van toestel en technologie.

Een eigenaardigheid is dat de minima voor de HTC One telkens lager liggen. Dit wordt vermoedelijk veroorzaakt door verschillende Android versies, met een verschillende aanpak op vlak van asynchrone scheduling.



Figuur 4.6: Laadtijd tiende resultaat liveboard in functie van toestel en technologie.

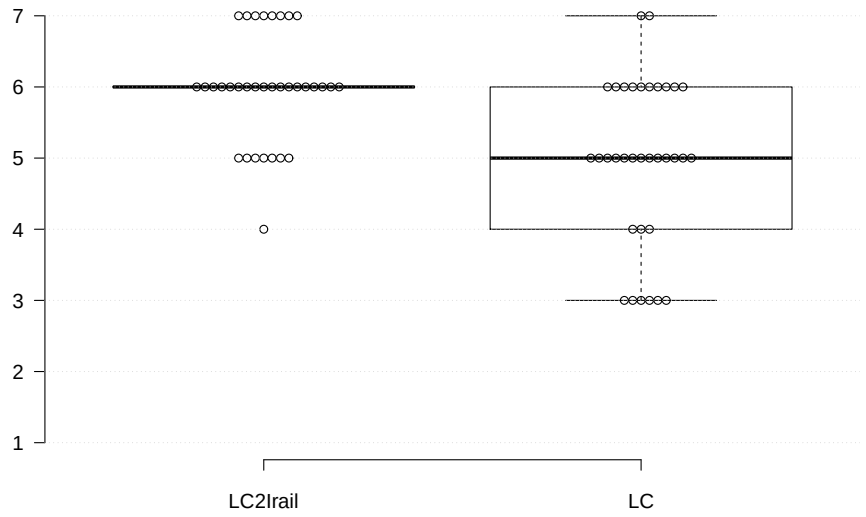
4.1.2 Ervaringen

Wanneer we naar de ervaringen van gebruikers gaan kijken, stemmen deze ongeveer overeen met wat we zouden verwachten na evaluatie van de metingen.

Zoals we in figuren 4.5 en 4.6 konden zien blijkt uit testen dat de performantie van LC2Irail consistenter is, zowel in de vorm van een kleinere spreiding van de resultaten op eenzelfde toestel, als in de vorm van kleinere verschillen tussen toestellen. Ook bij de gebruikerservaring zien we dit terugkomen. Wanneer de ervaren snelheid wordt uitgezet in een boxplot per techniek, zichtbaar in figuur 4.7, zien we net als bij de testen dat voor LC2Irail een heel consistente beoordeling wordt gegeven, terwijl deze voor Linked Connections veel meer uitgespreid, én iets lager ligt.

Hoewel 12 van de 17 testpersonen Linked Connections als redelijk tot extreem snel ervaart, zijn er slechts twee personen die Linked Connections sneller ervaren dan LC2Irail. De ervaringen en uiteindelijk keuze op basis van snelheid is zichtbaar in figuur 4.8. Het is duidelijk zichtbaar dat de ervaringen voor Linked Connections veel gemengder zijn dan de ervaringen voor LC2Irail. Er zijn zowel mensen die Linked Connections sneller, even snel of trager ervaren.

Alle testpersonen werden expliciet gevraagd welke implementatie ze als sneller ervoeren. Hierbij



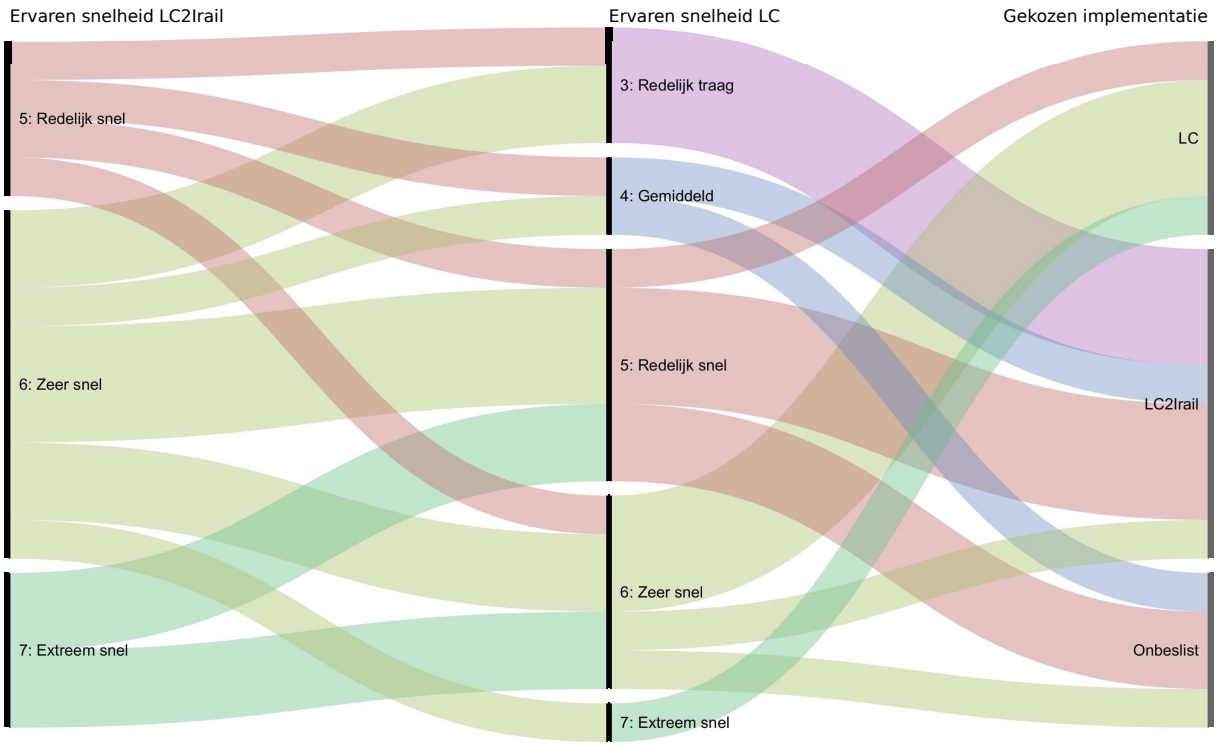
Figuur 4.7: De ervaren snelheid op een schaal 1-7 van vertrekken en aankomsten voor LC2Irail en Linked Connections, gebaseerd op 17 user-tests.

waren de antwoorden verdeeld: acht personen kozen de Linked Connections, vier personen hadden geen mening, en vijf personen kozen LC2Irail variant. Het is moeilijk om hier onmiddellijk conclusies uit te trekken. Om deze reden zullen we in hoofdstuk 5 de resultaten algemener bespreken. Wel kunnen we zeggen dat gebruikerservaringen zeer gemengd zijn voor Linked Connections, en zal Linked Connections sowieso niet voor de volledige populatie sneller zijn.

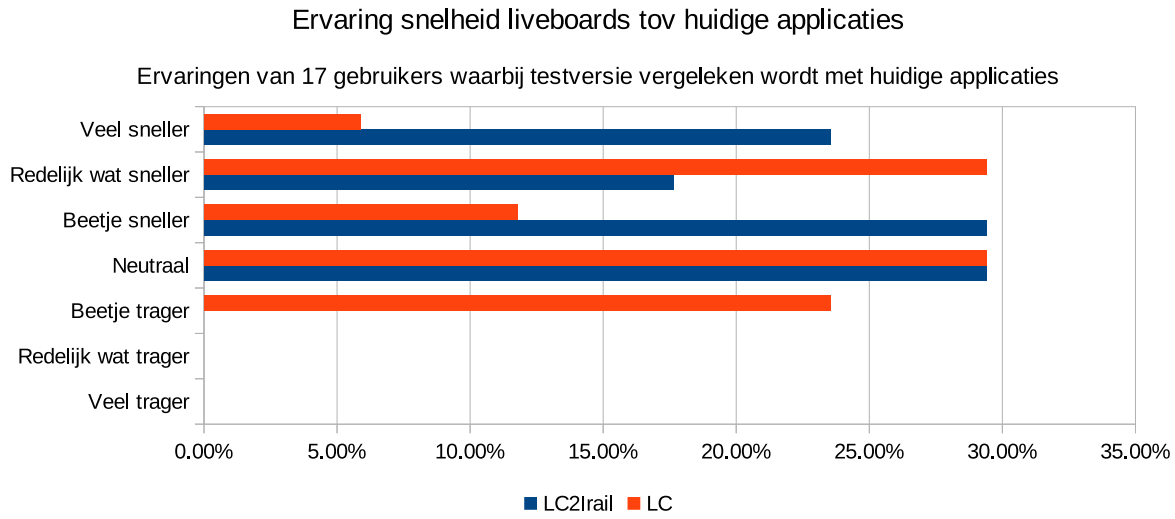
Wanneer we kijken naar de verschillen tussen de JSON parsers, blijkt dat beide parsers ongeveer even goed presteren in de ogen van de testers. Respectievelijk 5 op 7 en 7 op 10 testers zijn neutraal of tevreden, en bij beide varianten is er telkens een tester neutraal. Deze vergelijking is echter slechts een indicatie, en is door te kleine steekproeven ongeschikt om te veralgemenen naar een grotere populatie.

Wanneer echter gekeken wordt naar de gemeten prestatieverschillen tussen beide JSON parsers tijdens de usertests, zien we een duidelijk verschil, waarbij het 90e percentiel van de laadtijd onder de LoganSquare parser lager ligt dan de mediane laadtijd van de org.json parser. Dit verschil lijkt echter geen invloed te hebben op de ervaringen van gebruikers, vermoedelijk omdat men beide reeds als performant genoeg ervaart.

Als tot slot gevraagd wordt om de snelheid te vergelijken met de applicatie die de gebruiker op dit moment gebruikt, zichtbaar in figuur 4.9 , komen beide apps er goed uit. LC2Irail scoort hier zeer goed, en ondanks dat Linked Connections iets minder goed scoort dan LC2Irail, geven slechts 2 gebruikers aan Linked Connections trager te ervaren dan hun huidige applicatie.



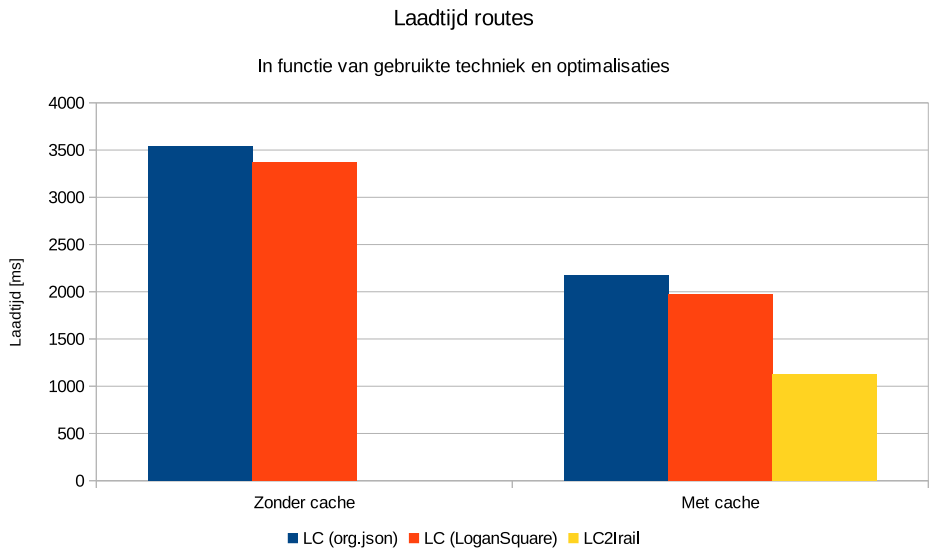
Figuur 4.8: Verbanden tussen de door gebruikers gekozen implementaties voor liveboards.



Figuur 4.9: De door 17 gebruikers ervaren snelheid liveboards ten opzichte huidige apps

4.2 Routes

4.2.1 Metingen

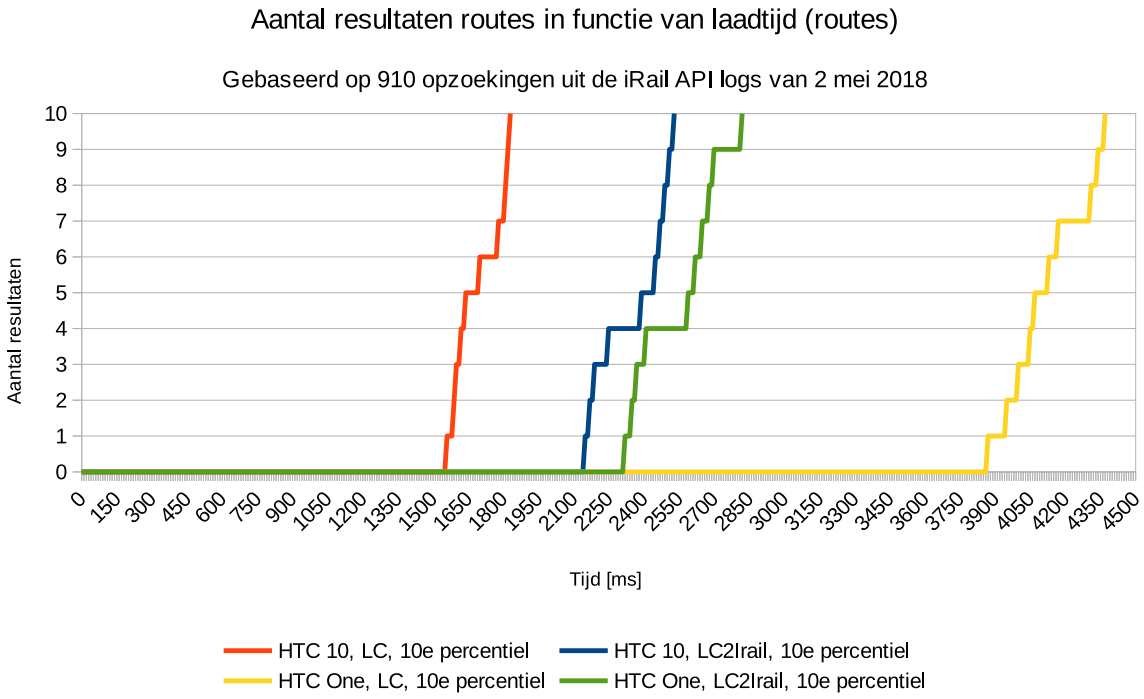


Figuur 4.10: De gemiddelde gemeten laadtijd voor routes gebruikmakend van een HTC 10 voor 779 opzoekingen gebaseerd op de iRail-logs.

Ook voor routes zullen we eerst kort de relatieve prestaties van verschillende implementaties bespreken. In grafiek 4.10 zijn de gemiddelde resultaten zichtbaar van een benchmark waarbij 779 routes opgezocht werden, ongeveer 5% van de opzoekingen door gebruikers op 2 mei 2018. Telkens is de minimale, gemiddelde en maximale responstijd gemeten, dit zowel gebruikmakend van de standaard JSON parser (*org.json*) en gebruikmakend van de *LoganSquare*-parser. Ook werd de test herhaald met cache in- en uitgeschakeld, om zo het effect hiervan te meten. Tot slot werd dezelfde test herhaald gebruikmakend van data afkomstig van de LC2Irail web applicatie om een vergelijking tussen de twee methodes te kunnen maken. Deze cijfers geven slechts een indicatie van de snelheid - een volledige en diepgaande statistische analyse van de performantieverschillen tussen verschillende implementaties van dezelfde techniek valt wegens tijdsgebrek buiten het bereik van deze masterproef.

Net zoals bij liveboards is ook hier de invloed van de cache duidelijk merkbaar. Dit valt te verklaren door de grote hoeveelheden data die verwerkt moeten worden, waarbij het cruciaal is dat deze niet steeds opnieuw gedownload wordt. Vergeleken met dezelfde analyse voor Liveboards (figuur 4.1), zien we hier een minder groot verschil tussen de parsers. Dit komt mogelijk door de grotere impact van de verdere algoritmes, waardoor de invloed van parsing verkleint. Het algoritme om de data tot routes te verwerken is het zwaarst van de drie endpoints.

Om ook hier een exact beeld te vormen van de prestaties, maken we ook hier een duizendtal opzoekingen. Hiervoor kiezen we telkens de vijfde opzoeking uit de iRail-logs. Voor elke route wordt gepoogd 10 resultaten te laden. De resultaten hiervan zijn zichtbaar in grafieken 4.11, 4.12 en 4.13, respectievelijk voor het tiende, vijftigste en negentigste percentiel.



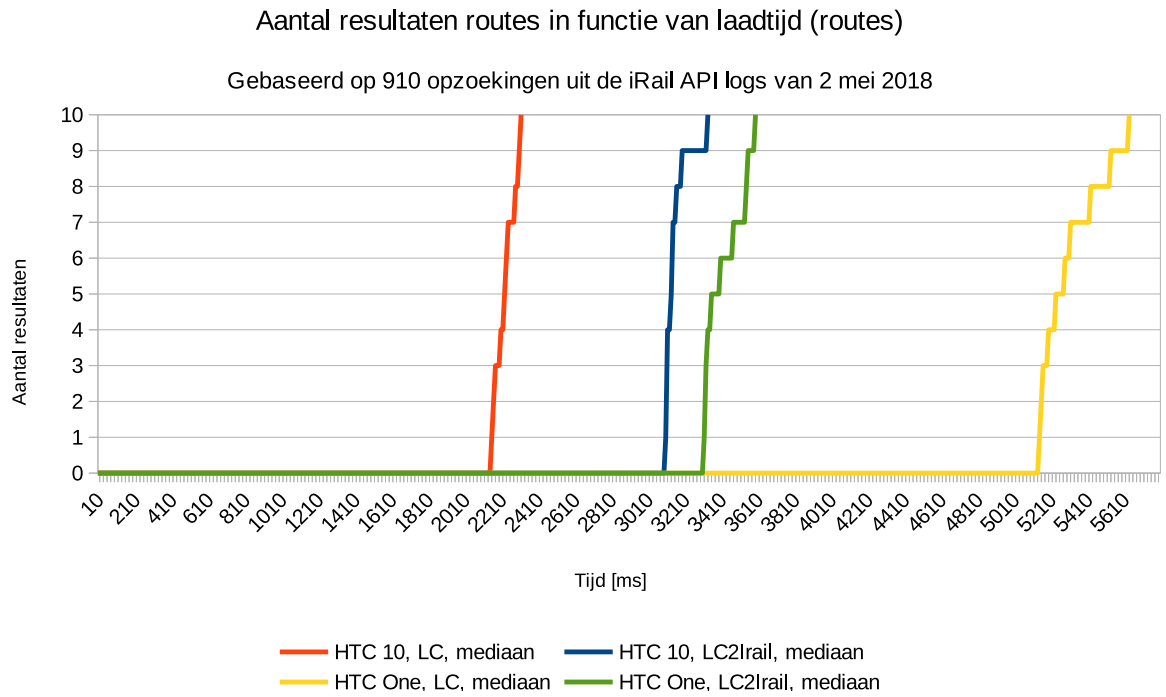
Figuur 4.11: Het aantal resultaten in functie van de verlopen tijd.

Uit deze grafieken kunnen we opnieuw duidelijke conclusies trekken:

- In alle grafieken en voor alle testen, hebben de curves een gelijkaardige vorm, waarbij er na een relatief lange wachttijd aan snel tempo resultaten geladen worden: in het geval van Linked Connections is voor de eerste opzoeking telkens een relatief grote hoeveelheid data nodig is, waarna slechts één of twee extra pagina's moeten opgehaald worden om het volgend resultaat te bepalen. In het geval van LC2Irail worden resultaten in grote blokken binnengehaald, waarbij vanaf de tweede opzoeking reeds veel data in cache zitten. In het geval van LC2Irail worden resultaten ook onmiddellijk voor grote intervallen opgehaald, om zo het aantal verzoeken te beperken.

- Terwijl op de HTC 10 Linked Connections in alle gevallen beter presteert dan LC2Irail, presteert Linked Connections slechter dan LC2Irail op de HTC One.

- Opnieuw presteert LC2Irail op beide toestellen gelijkaardig, met slechts een kleine verschuiving in tijd tussen beide curves.

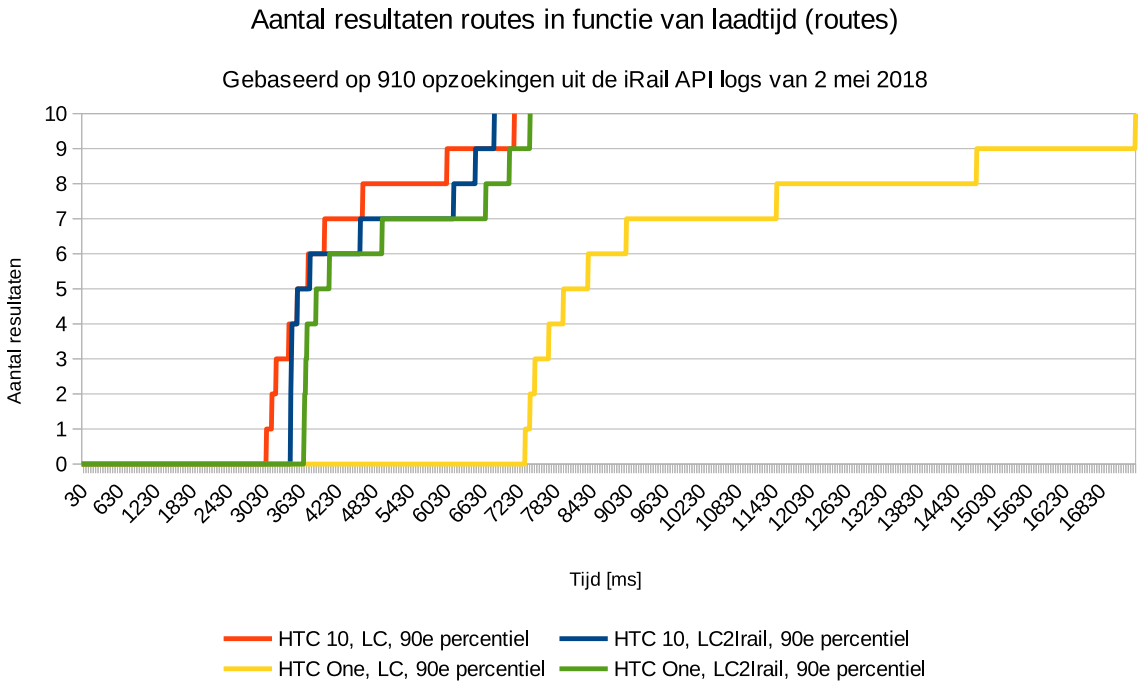


Figuur 4.12: Het aantal resultaten in functie van de verlopen tijd.

- Terwijl in het slechtste geval bijna alle varianten gelijk presteren, loopt Linked Connections op de HTC One enorm achter. Uit alle grafieken volgt dat hoe trager het toestel, hoe trager Linked Connections, terwijl LC2Irail ongeveer even goed blijft presteren. Bijgevolg kunnen we dus ook stellen, dat alle toestellen trager dan de HTC10 in het slechtste geval trager zullen presteren dan de HTC One.
- In alle gevallen laadt het eerste resultaat pas na anderhalve tot zeven seconden. Dit zijn zeer lange laadtijden, waarvan we verwachten dat ze de gebruikerservaring negatief gaan beïnvloeden.

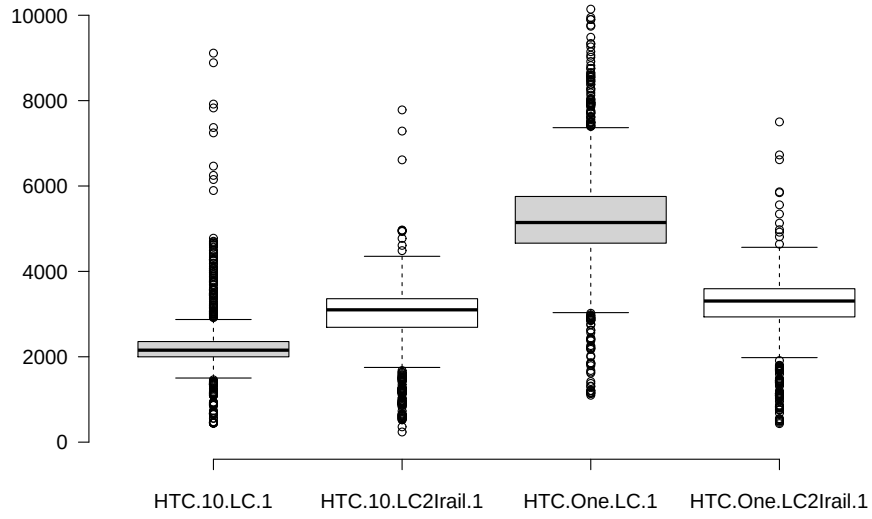
Wanneer we specifiek naar de verdelingen kijken, gevisualiseerd door middel van de boxplots in figuur 4.14 en 4.15, zien we sterke verschillen, zowel tussen toestellen als implementaties. Net als bij liveboards zien we ook hier duidelijk hoe LC2Irail gelijke prestaties heeft op beide toestellen, met bijna identieke distributies. Dit in tegenstelling tot de prestaties van Linked Connections, die zeer sterk variëren per toestel. Op de HTC 10 zal voor al meer dan 75% van de opzoekingen via Linked Connections het eerste resultaat geladen zijn op het moment dat LC2Irail op hetzelfde toestel minder dan 25% van de verzoeken beantwoord heeft. Op het HTC One toestel is dit echter omgekeerd, en nog extremer.

Voor het tiende resultaat zijn deze verschillen tussen toestellen iets minder extreem, al zijn ze nog steeds zeer groot. Linked Connections blijft iets consistentere resultaten bieden op de HTC

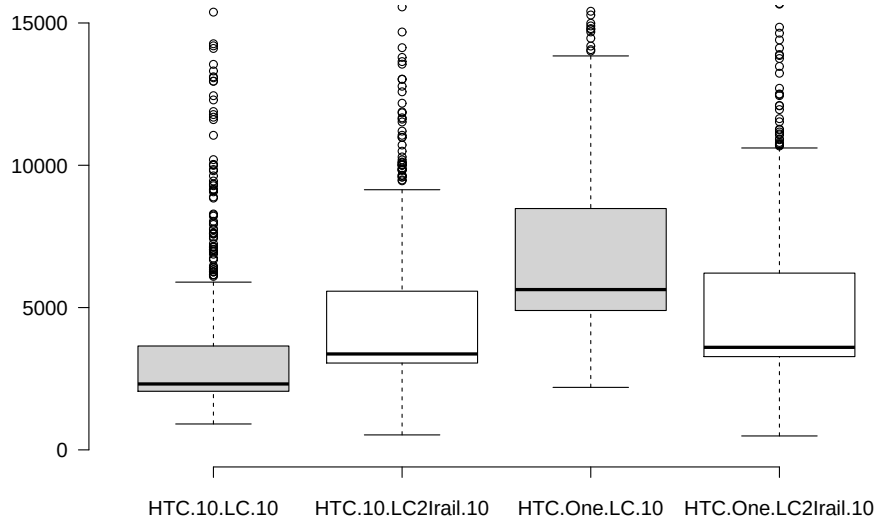


Figuur 4.13: Het aantal resultaten in functie van de verlopen tijd.

10, terwijl de HTC One net minder consistente resultaten biedt dan LC2Irail. LC2Irail biedt ook voor het tiende resultaat exact dezelfde ervaring op beide toestellen, terwijl het verschil tussen de mediaan voor Linked Connections op beide toestellen niet acceptabel is en de gebruikerservaring duidelijk zal beïnvloeden.



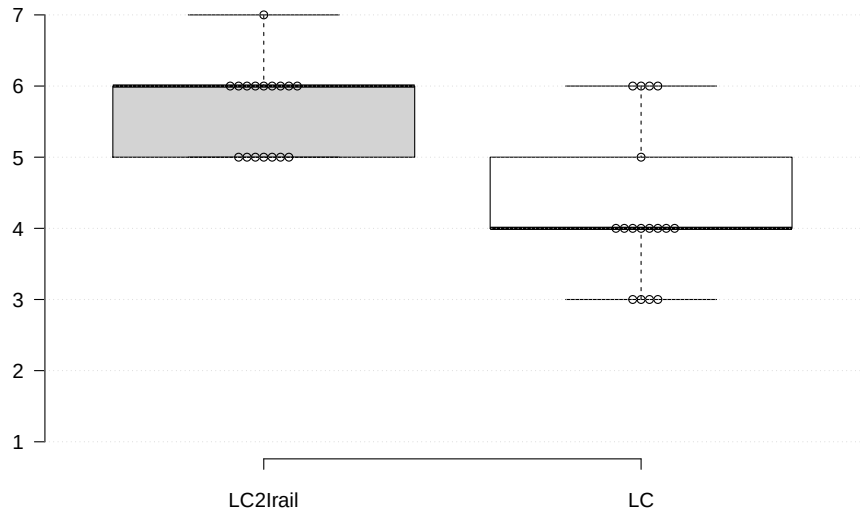
Figuur 4.14: Laadtijd eerste resultaat route in functie van toestel en technologie.



Figuur 4.15: Laadtijd tiende resultaat route in functie van toestel en technologie.

4.2.2 Ervaringen

Op vlak van gebruikerservaring verwachten we dat gebruikers net zoals bij Liveboards de implementatie op basis van LC2Irail consistenter zullen beoordelen, en dat, afhankelijk van het door de tester gebruikte toestel, Linked Connections sneller, even snel of trager dan LC2Irail ervaren wordt.

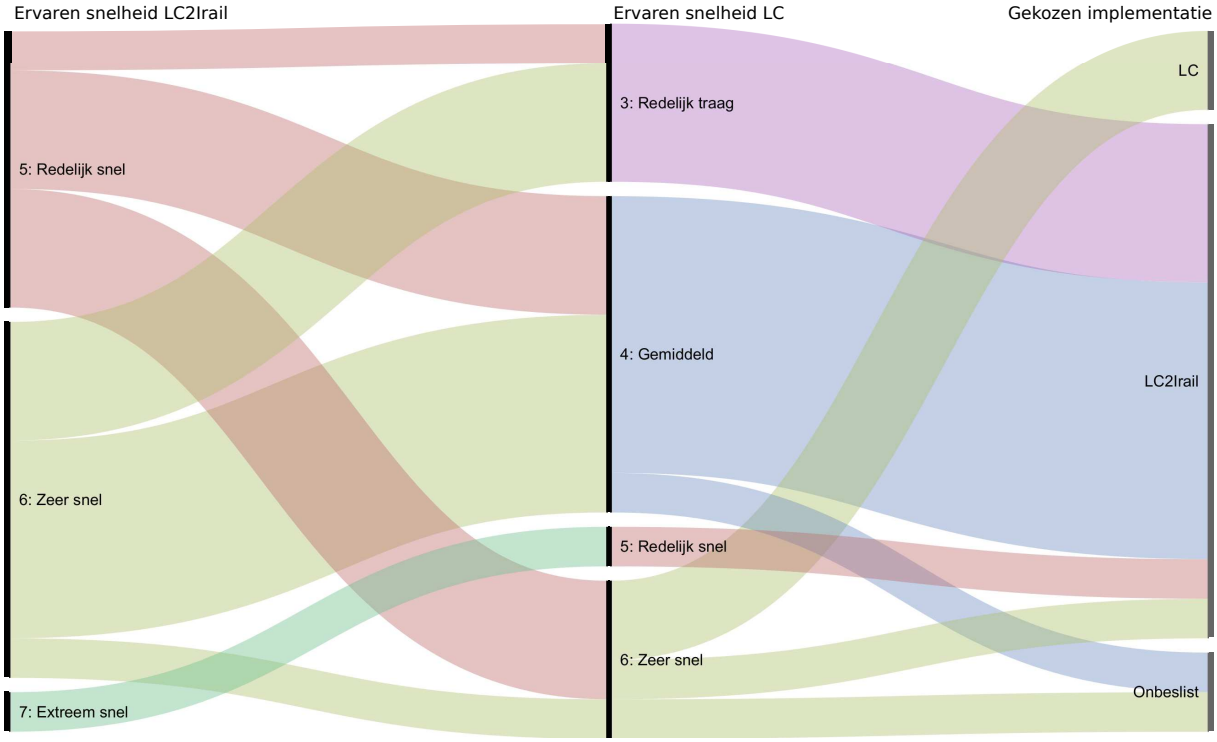


Figuur 4.16: De ervaren snelheid op een schaal 1-7 van routes voor LC2Irail en Linked Connections, gebaseerd op 17 user-tests.

Wanneer we de resultaten van user-testing vergelijken met de verwachtingen, blijken deze verwachtingen grotendeels in vervulling te gaan. In figuur 4.16 is te zien dat de prestaties van LC2Irail iets consistenter goed beoordeeld worden, en LC2Irail een betere beoordeling krijgt dan Linked Connections. In vergelijking met liveboards (figuur 4.7) zien we dat gebruikers bij routes iets minder verdeeld zijn over de prestaties van Linked Connections, al komt dit omdat er geen gebruikers voor de hoogste score kiezen.

De consistentere beoordelingen voor Linked Connections zijn ook duidelijk wanneer we dieper ingaan op hoe eenzelfde gebruiker de snelheid van LC2Irail en Linked Connections ervaart, zichtbaar in figuur 4.17. Hoewel veel gebruikers Linked Connections als trager ervaren, is er ook een klein aantal gebruikers dat Linked Connections sneller ervaart. Het zijn deze personen die, wanneer expliciet gevraagd wordt om de snelste implementatie aan te duiden, voor Linked Connections kiezen. Dit duidelijk verband kon bij liveboards niet gelegd worden (figuur 4.8), wat te wijten kan zijn aan een voor gebruikers slechts klein, onduidelijk verschil tussen de prestaties bij liveboards. Voor routes wordt het verschil veel duidelijker ervaren, en kiezen gebruikers dus

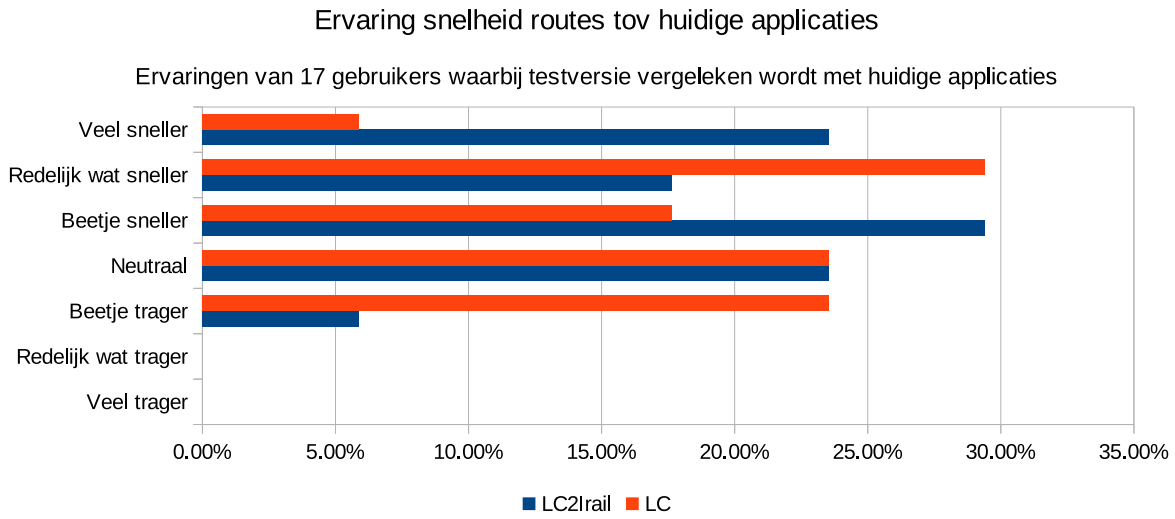
consistenter met de door hun ervaren snelheden. Opvallend is ook dat er een persoon is die Linked Connections als zeer snel bestempelt, en Linked Connections hiermee even snel of sneller dan LC2Irail ervaart. Echter kiest deze persoon toch voor LC2Irail wanneer een expliciete keuze gemaakt dient te worden. We kunnen stellen dat voor deze gebruiker het verschil niet merkbaar was. Aan de andere kant kiest iedereen die Linked Connections als traag of gemiddeld ervaart voor LC2Irail, op één gebruiker na die onbeslist is. Voor deze gebruikers moet Linked Connections nog sneller gemaakt worden alvorens het concurrentieel wordt.



Figuur 4.17: Verbanden tussen de door gebruikers gekozen implementaties voor routes.

Wanneer we voor routes beide JSON parsers vergelijken, zien we dat voor de *LoganSquare*-parser de proefpersonen een meer uitgesproken mening hadden: er waren zowel meer tevreden als ontevreden personen, terwijl bij de *org.json* parser veel mensen neutraal waren. Dit gaat echter in tegen een praktijktest waarbij enkele gebruikers achtereenvolgens een versie gebruikmakend van de *org.json* en *LoganSquare*-parser voorgeschoteld kregen, gaven deze telkens aan de versie op basis van *LoganSquare* sneller te ervaren, zowel op goedkope als dure smartphones. Hieruit besluiten we dat de gebruikerstests, opgedeeld per parser, te kleine steekproeven zijn om een algemene conclusie te vormen over de invloed van de parsers.

Als tot slot gevraagd wordt om de snelheid te vergelijken met de applicatie die de gebruiker op dit moment gebruikt, doen beide testversies het goed ten opzichte van de huidige applicaties. Dit is zichtbaar in figuur 4.18. Linked Connections scoort iets minder goed dan LC2Irail, maar de overgrote meerderheid vindt Linked Connections nog steeds even snel of sneller dan de applicatie



Figuur 4.18: De door 17 gebruikers ervaren snelheid routes ten opzichte huidige apps

die men gewoonlijk gebruikt.

4.3 Voertuigen

4.3.1 Metingen

Het opzoeken van het traject dat een voertuig aflegt verschilt sterk van de eerder besproken endpoints. Het traject van een voertuig wordt door HyperRail als één (ondeelbaar) resultaat beschouwd, en kan hierdoor niet per stop geladen worden. Dit in tegenstelling tot liveboards en routes, waar er verschillende resultaten zijn die elk onafhankelijk van elkaar zijn.

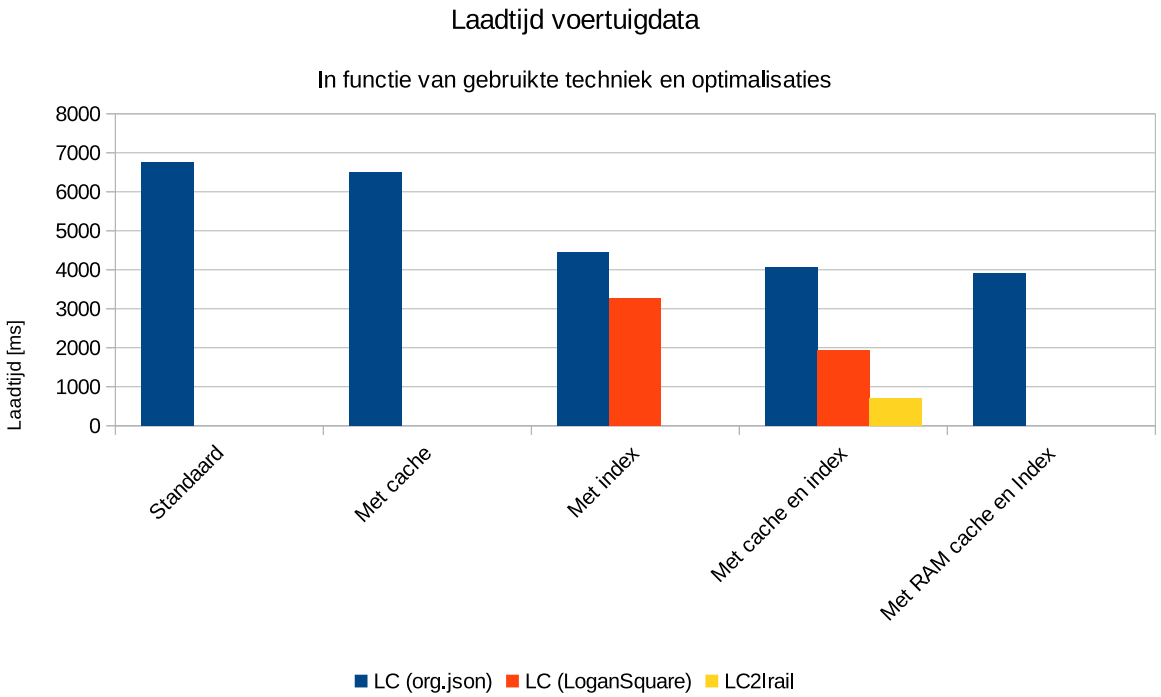
Dit is ook de opzoeking die het meeste data vereist bij Linked Connections: alle pagina's moeten doorzocht worden op connecties met betrekking tot één specifiek voertuig. Dit voertuig komt slechts in een relatief beperkt aantal pagina's voor, gezien het voertuig slechts enkele uren rijdt, en het tijdstip van vertrek en aankomst onbekend zijn. Zoals eerder vermeld zijn hier enkele oplossingen voor, zoals het gebruik van een index. We definiëren een index in deze context als een lijst van alle treinen voor een bepaalde periode (in dit geval mei 2018) en het tijdstip van hun eerste vertrek.

Om een idee te krijgen van de invloed van deze index, alsook van het gebruik van een cachegeheugen voor de Linked Connections pagina's bij deze opzoekingen, werden 102 voertuigen opgezocht, voor alle combinaties van cache en index gebruik. Tevens werd een extra test gedaan met een cache die in het RAM-geheugen geplaatst wordt (in tegenstelling tot het flashgeheugen van het toestel), en een vergelijkende test waarbij de Linked Connections server gebruikt werd. De gemiddelde opzoektijd hiervoor is gevisualiseerd in figuur 4.19.

Het is duidelijk dat de standaard implementatie zeer slecht presteert. Ook het gebruik van een cachegeheugen brengt hierbij niet veel beterschap. Wanneer echter een index toegevoegd wordt, is een drastische verbetering merkbaar. Het gemiddelde daalt in deze beperkte test met ongeveer een derde. Toevoeging van een cachegeheugen, op flash of in het RAM-geheugen, brengt ook hier relatief weinig beterschap.

Een tweede grote verbetering kan behaald worden door het gebruik van de eerder besproken *LoganSquare*-parser. Hierbij zien we ook een veel grotere verbetering door cachegebruik dan bij de *org.json* parser. Dit is logisch, gezien bij het gebruik van de *LoganSquare*-parser het verwerken van de data relatief gezien minder tijd in beslag neemt - het ophalen van data wordt dus belangrijker. Op het eerste zicht blijven alle lokale varianten veel trager dan de serverimplementatie, die sneller door pagina's kan zoeken.

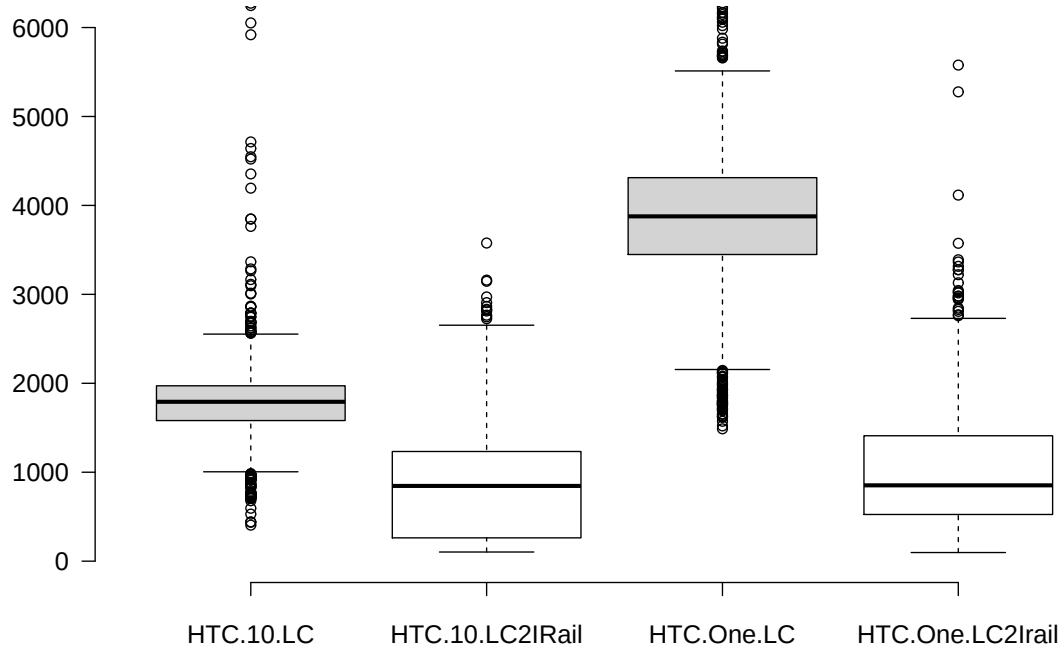
We onderzoeken nu het verschil tussen de lokale implementatie en de serverimplementatie in detail. Hiervoor zoeken we 2102 ritten op die plaatsvinden op 6 mei 2018. Dit wordt enerzijds gedaan voor de Linked Connections implementatie die gebruik maakt van de *LoganSquare*-parser,



Figuur 4.19: De gemeten laadtijd voor voertuigen gebruikmakend van een HTC 10 voor 102 opzoekingen gebaseerd op de iRail-logs.

cache en lokale index, en anderzijds voor de serverimplementatie, die server-side over dezelfde index en een cache beschikt.

Wanneer we kijken naar de boxplot van de responstijd, weergegeven in figuur 4.20, zien we dat de Linked Connections duidelijk slechter presteert. Op beide toestellen is LC2Irail sneller. Bij de HTC 10, valt dit nog enigszins mee, maar op de HTC One zijn via LC2Irail de meeste resultaten binnen 3000 milliseconden geladen, terwijl op dat moment nog geen 25% van de opzoekingen via Linked Connections uitgevoerd werd. Net zoals bij liveboards en routes zien we hier dat LC2Irail consistente prestaties biedt: beide boxplots zijn ongeveer identiek, op wat uitlopers na. Voor Linked Connections zien we echter dat, net zoals voor het opzoeken van liveboards en routes, de spreiding van de benodigde tijd afhangt van het toestel: een traag toestel zal niet alleen trager resultaten laden, maar heeft ook een grotere variatie in de laadtijd.

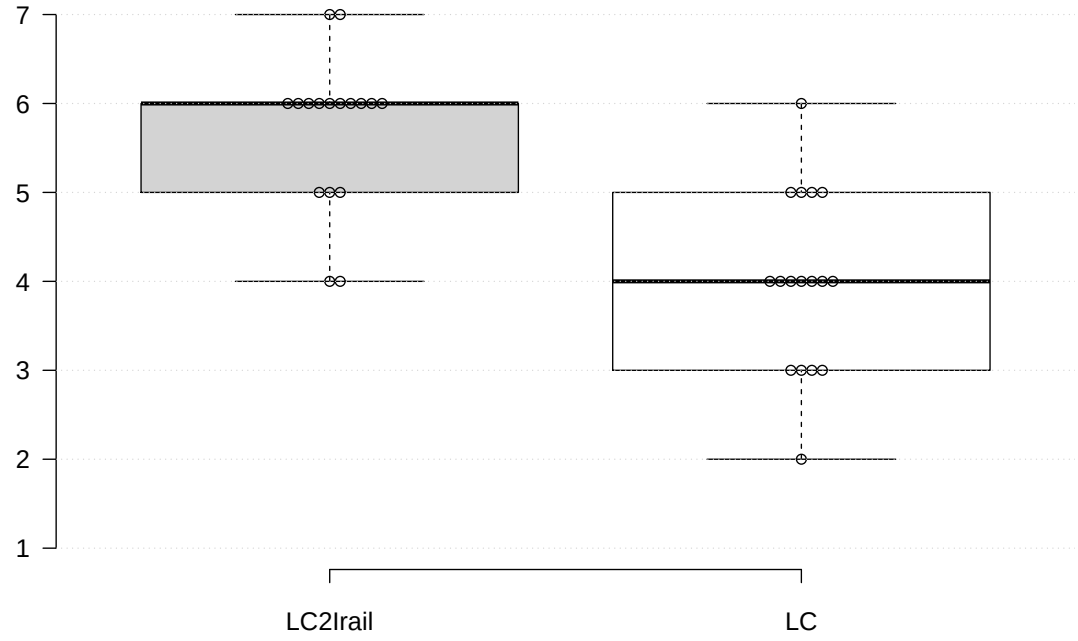


Figuur 4.20: De prestaties voor het laden van voertuigen, gemeten door alle voertuigen, beschreven in Linked Connections, voor 6 mei op te zoeken.

4.3.2 Ervaringen

Wanneer we de resultaten van de user-testing bekijken, zien we zoals verwacht dat het laden van voertuigen beduidend slechter scoort wanneer de lokale Linked Connections implementatie gebruikt wordt, vergeleken met wanneer de serverimplementatie gebruikt wordt. In figuur 4.20 is dit duidelijk zichtbaar. Zo beoordelen de meeste gebruikers Linked connections slechts als ”gemiddeld”, terwijl de meerderheid van de gebruikers de LC2Irail variant als ’Zeer snel‘ bestempelde. Ook zien we hier, net als bij liveboards en routes, dat er voor Linked Connections een veel grotere spreiding is in de gegeven antwoorden, terwijl bij LC2Irail iedereen het er over eens lijkt dat deze implementatie snel is.

Wanneer we de testgroep van Linked Connections opsplitsen per gebruikte parser, blijkt dat personen die de lokale implementatie op basis van *LoganSquare* testten, de laadtijd iets beter beoordeelden vergeleken met de groep die de implementatie op basis van de *org.json* parser testte. Deze resultaten liggen ook in lijn met ervaringen van testers die beide parsers achtereenvolgens voorgeschoteld kregen, waarbij alle testers de *LoganSqare* parser sneller ervoeren. Ondanks dat de testgroep onvoldoende groot was om een veralgemening te kunnen maken, kunnen we in

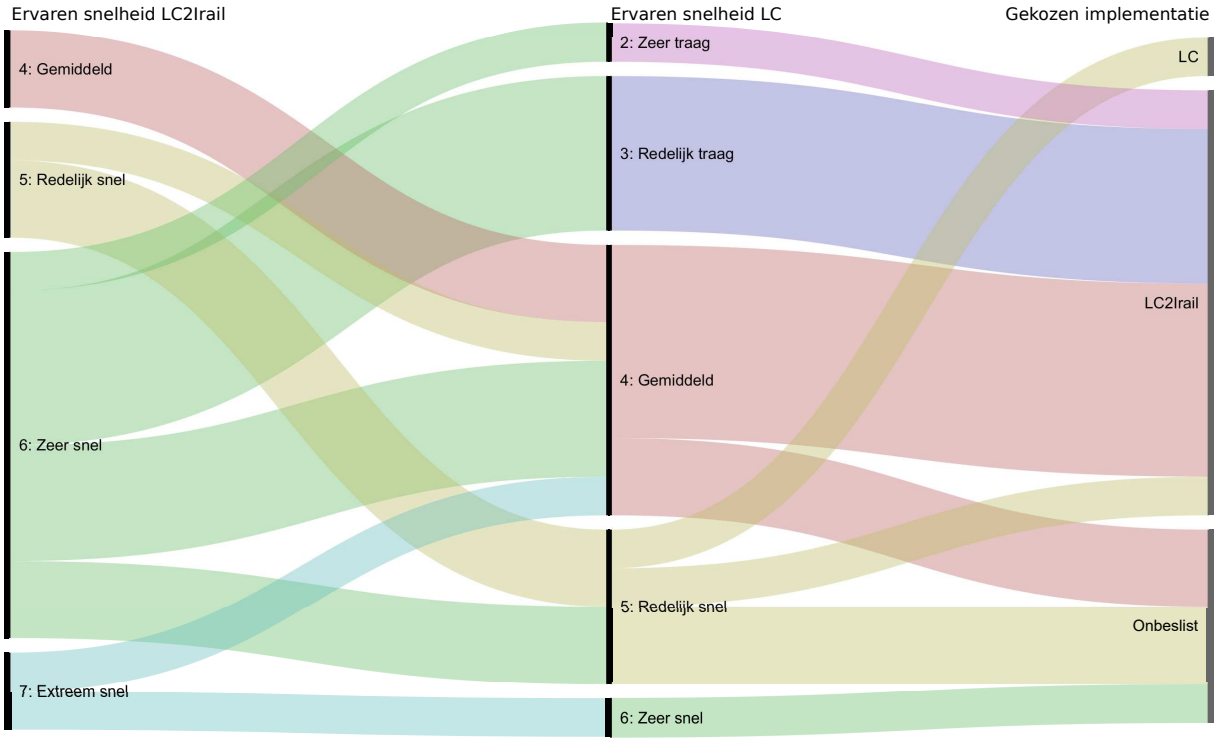


Figuur 4.21: De ervaren snelheid op een schaal 1-7 van routes voor LC2Irail en Linked Connections, gebaseerd op 17 user-tests.

combinatie met de directe vergelijking wel stellen dat er een grote kans is dat verbeteringen in de implementatie de snelheid verder omlaag kunnen brengen, en zo de gebruikerservaring kunnen verbeteren. Gezien bij het berekenen van voertuigen het meeste data nodig is, is hier de impact van implementatiedetails het grootst.

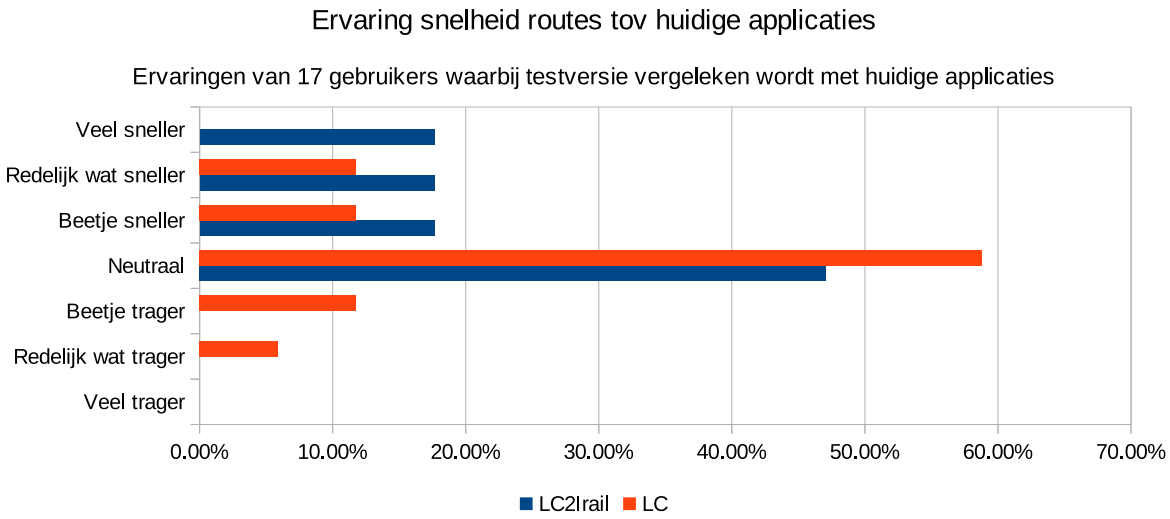
Wanneer de gebruiker gevraagd werd te kiezen, koos slechts één gebruiker voor de lokale implementatie in dit onderdeel. Vijf gebruikers hadden geen specifieke voorkeur voor een specifieke implementatie, ook al beoordeelden vier van hen Linked Connections als trager. In figuur 4.22 zijn de ervaringen van elke gebruiker duidelijk te zien. Zo zien we dat de ervaring voor gebruikers nooit verbeterd, en veel gebruikers een groot verschil ervaren tussen de snelheid van beide implementaties, in het nadeel van Linked Connections. Veel gebruikers die Linked Connections als redelijk of zeer snel ervaren, ervoeren LC2Irail nog steeds als sneller, waardoor ze wanneer ze moesten kiezen niet voor Linked Connections kozen. De enigste gebruiker die voor Linked Connections koos, ervoer Linked Connections niet als sneller dan LC2Irail, maar vond beide wel snel.

Als gevraagd wordt om de snelheid te vergelijken met de applicatie die de gebruiker op dit moment gebruikt, geven zes op tien gebruikers aan Linked Connections als even snel te ervaren



Figuur 4.22: Verbanden tussen de door gebruikers gekozen implementaties voor voertuigen.

als hun huidige applicatie voor het opzoeken van voertuigen. De andere gebruikers geven aan de opzoekingen redelijk wat trager tot redelijk wat sneller te ervaren. Dit is duidelijk zichtbaar in figuur 4.23. Terwijl LC2Irail acceptabel scoort, blijkt Linked Connections hier toch achterop te raken, zowel ten opzichte van LC2Irail als ten opzichte van de Linked Connections prestaties voor voertuigen (figuur 4.9) en routes (figuur 4.18).



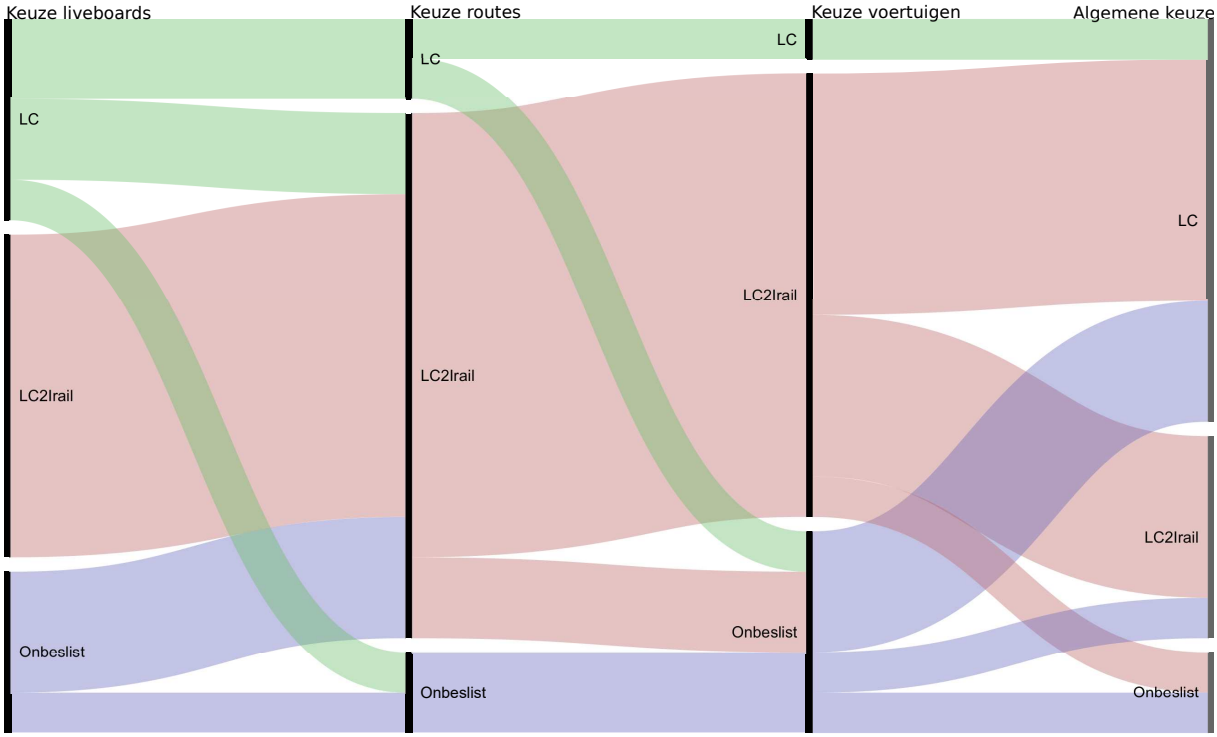
Figuur 4.23: De door 17 gebruikers ervaren snelheid routes ten opzichte huidige apps

4.4 Door de gebruiker gekozen implementatie

Voor alle soorten informatie (liveboards, routes, en voertuigen) lijkt Linked Connections een gelijkaardige of slechtere gebruikerservaring op te leveren dan LC2Irail in termen van laadtijd. Hierbij dienen we op te merken dat dit verschil bij liveboards slechts zeer beperkt is, en het laden nog steeds als snel werd ervaren. Voor routes bestempelden enkele personen Linked Connections als traag, maar ook hier blijft het verschil beperkt. Bij voertuigen blijkt echter dat Linked Connections door drie kwart van de gebruikers als trager werd ervaren, waarbij Linked Connections niet enkel relatief slechter scoort, maar ook in absolute termen slechts door een minderheid van de gebruikers als snel wordt ervaren.

Linked Connections verschilt echter niet enkel in termen van laadtijd. Zoals eerder aangehaald in hypothese 1 is offline opzoeken mogelijk, en vermoeden we dat dit de keuze van de gebruiker beïnvloedt. Om dit na te gaan hebben we de keuzes van gebruikers gevisualiseerd in figuur 4.24. In dit diagram zien we zowel hoeveel gebruikers voor elke implementatie kozen, maar ook hoe de keuze van gebruikers evolueert. Zo zien we dat naarmate de relatieve prestaties van LC ten opzichte van LC2Irail dalen, personen die eerder voor Linked Connections kozen overstappen op LC2Irail.

Terwijl de meerderheid van de gebruikers steeds voor LC2Irail koos, kantelt deze balans volledig om wanneer gebruikers worden gevraagd om met alle aspecten rekening te houden. Dit is duidelijk zichtbaar aan de rechterkant van figuur 4.24. Hieruit blijkt dat gebruikers enige snelheid willen opgeven in ruil voor offline opzoekingen. Zeven gebruikers laten weten dat een hybride systeem ideaal zou zijn, waarbij de snelheid van LC2Irail gecombineerd wordt met Linked



Figuur 4.24: Verbanden tussen de door gebruikers gekozen implementaties voor alle soorten informatie, alsook de resulterende keuze waarbij ook offline toegang in rekening werd gebracht.

Connections als offline alternatief. Wanneer deze gebruikers alsnog verplicht werden te kiezen, waren hun keuzes gelijk verdeeld, afhankelijk van de persoonlijke nood om offline te kunnen opzoeken.

We trachten nu een antwoord te vinden op de in hoofdstuk 3 gestelde vragen omtrent de gebruikerservaring.

- Ervaart de gebruiker een app die lokaal Linked Connections gebruikt als sneller dan een app die gebruik maakt van een RPC API?
  De ervaring van de gebruiker hangt sterk af van het gebruikte toestel en de gemaakte opzoekingen. Enkel gebruikers van snelle toestellen ervaren Linked Connections in sommige gevallen als sneller. Voor liveboards is Linked Connections concurrentieel, maar voor andere opzoekingen is deze techniek meestal merkbaar trager dan de gebruikte RPC API.
- Ervaart de gebruiker een app die lokaal Linked Connections gebruikt als sneller dan zijn huidige app?
  Een minderheid ervaart Linked Connections als sneller dan de huidige gebruikte applicatie. Voor liveboards, routes en voertuigen vinden respectievelijk 45%, 30% en 24% dat Linked Connections sneller is. Respectievelijk 30%, 40% en 60% van de gebruikers ervaren Linked Connections als even snel, met 24%, 30% en 12% van de gebruikers die Linked Connections

expliciet als trager ervaren dan hun huidige applicatie. Linked Connections brengt op dit moment zeker geen verbetering in snelheid voor de gebruikers.

Wat dit wil zeggen voor de onderzoeksvraag en hypotheses en zullen we verder bespreken in hoofdstuk 5.

4.5 Enquête

Zoals vermeld in hoofdstuk 3, trachten we een aantal vragen te beantwoorden aan de hand van een enquête (bijlage **??**). We zullen nu eerst de resultaten van deze enquête overlopen, alvorens aan de hand van deze resultaten een antwoord te zoeken op de in hoofdstuk 3 geformuleerde vragen.

4.5.1 Antwoorden van respondenten

Er werden in totaal *81 volledig ingevulde enquêtes* digitaal verzameld, in een gevarieerd publiek. Zo neemt meer dan de helft van de respondenten meerdere keren per week (26%) of dagelijks (28%) de trein. Ook personen die occasioneel reizen zijn goed vertegenwoordigd, zo reist 16% minder dan één keer per maand per trein.

Wanneer we kijken naar de *informatiebronnen* voor reizigers, blijkt dat *native* applicaties voorop staan (93%, waarvan 27% third-party applicaties zijn), gevolgd door digitale informatieborden en affiches (74%) en websites (62%) . Hierbij moeten we opmerken dat de enquête specifiek gericht is op personen die applicaties gebruiken om informatie op te zoeken, en het werkelijk aandeel van de reizigers die applicaties gebruikt dus iets lager kan liggen.

Voor iets minder dan de helft (48%) van de bevraagde reizigers verloopt elke reis probleemloos. Alle respondenten geven aan vertragingen te ervaren bij het reizen, waarbij 33% aangeeft dat dit zelfs meestal of altijd het geval is. Na vertragingen zijn spoorwijzigingen de tweede grootste bron van ergernis: meer dan 85% van de gebruikers ervaart dit wel eens, waarbij dit voor 15% van de reizigers regelmatig voorvalt. Tot slot geeft 72% van de respondenten aan wel eens een afgeschafte trein te hebben, al gebeurt dit voor slechts 1% de helft van de tijd.

Wanneer we nagaan hoe *up-to-date informatie* voor reizigers is, wordt duidelijk dat hier zeker ruimte is voor verbetering: slechts 25% zegt altijd over actuele informatie in de stations te beschikken, applicaties doen het iets beter, waarbij 40% van de gebruikers altijd over actuele informatie beschikt. Voor beide blijkt dat 50% van de personen die dit ervaart, er slechts soms last van heeft. Echter blijkt wel dat 27% van de personen minstens de helt van de tijd dit probleem in

stations ervaart. Applicaties doen het iets beter, waar slechts 10% van de gebruikers dit probleem minstens de helft van de tijd ervaart. Uit deze cijfers kunnen we concluderen dat gebruikers wel degelijk nood hebben aan actuele informatie.

Applicaties blijken ook de *voornaamste bron van informatie* te zijn voor gebruikers: Voor alle problemen, op spoorwijzigingen na, checken gebruikers hun smartphone. Voor spoorwijzigingen blijft informatie in de stations zelf, zoals omgeroepen informatie of digitale borden populairder. Ook wanneer informatie in de applicatie niet actueel is zoeken mensen hun toevlucht tot de omgeroepen informatie of digitale borden. Wanneer gebruikers gevraagd worden om informatiebronnen naar gebruik te rangschikken, blijven applicaties en websites ook hier bovenaan staan.

Wanneer we gaan kijken naar de *tevredenheid*, blijkt dat applicaties hier uitzonderlijk goed scoren: meer dan 77% van de gebruikers is hierover tevreden. Dit vormt een scherp contrast met websites, waar slechts 43% tevreden over is. Voor alle informatiebronnen zijn er ontevreden gebruikers, wat er opnieuw op wijst dat er ruimte is voor verbetering.

Bij de respondenten zijn Android en iOS gelijk vertegenwoordigd, met respectievelijk 39 en 40 respondenten. Een enkele respondent gebruikt Windows Mobile, nog een enkeling gebruikt Sailfish OS. 73% gebruikt voornamelijk de officiële NMBS applicatie, de andere 27% is ongeveer gelijk verdeeld over third-party applicaties, zoals HyperRail, iRail, Railer en BeTrains (tesamen goed voor 21%) en een mix van algemene en buitenlandse applicaties, zoals citymapper, de Lijn en Deutsche Bahn. Het aandeel van third-party applicaties kan in dit onderzoek mogelijk beïnvloed zijn, gezien de enquête onder andere verspreid werd via distributiekanalen gelinkt aan iRail en deze third-party applicaties.

Deze applicaties gebruiken we *vooral in stations* (95% van de gebruikers), maar ook thuis (85%) en op de trein zelf (80%). Op de trein zijn we echter niet tevreden over de snelheid waarmee resultaten laden (66% tevreden), thuis zijn we iets tevredener (72% tevreden). Ook de gebruiksvriendelijkheid van opzoekingen daalt tijdens het reizen.

Naar mogelijke oorzaken van deze dalingen tijdens een reis hoeven we niet ver te zoeken: 60% van de reizigers is ontevreden over het mobiele netwerk tijdens een treinreis. Maar liefst 96% van de reizigers heeft last van traag of niet ladende webpagina's, waarvan de meerderheid hier hier minstens de helft van de tijd hinder door ondervindt. Diezelfde mobiele data is voor 53% van de respondenten ook een bron van angst - ondanks dat er steeds meer data bij abonnementen geleverd wordt, blijven we schrik hebben om te veel data te verbruiken. Hier dienen we wel onmiddellijk te nuanceren: 21% maakt zich slechts een beetje zorgen. Deze groep zal vermoedelijk vooral voorzichtig zijn met media, en niet zozeer met het gebruik van routeplanning applicaties. Vooral jongeren (jonger dan 18) en personen ouder dan 35 jaar zijn voorzichtig met hun mobiele data.

Wanneer we *informatie niet opzoeken via de app*, is dit voornamelijk omdat de gebruiker geen mobiele data heeft (33%), omdat het opzoeken te lang duurt (24%), of omdat informatie in het station handiger is. We maken ons iets meer zorgen om het batterijgebruik van de applicatie (19%) dan om het dataverbruik (15%).

Wanneer gebruikers gevraagd worden naar *wat ze belangrijk vinden in een applicatie* voor openbaar vervoer, komt het snel laden van resultaten overduidelijk op de eerste plaats. Dit wordt gevolgd door offline zoekopdrachten, waarna privacy, batterijgebruik en dataverbruik kort op elkaar volgen.

Ondanks dat *privacy* een hot topic is, geeft 53% van de gebruikers aan zich hier geen zorgen om te maken. Dit zouden we misschien beter wel doen, want 75% weet niet zeker of zijn of haar reisplannen over internet verstuurd worden, terwijl alle applicaties dit op dit moment doen. 12% is er zelfs zeker van overtuigd dat zijn of haar reisplannen niet over internet verzonden worden. Ook over het versturen van onze locatie zijn we slecht geïnformeerd. Zo meent 17% onterecht dat zijn of haar exacte locatie waarschijnlijk niet over internet verstuurd wordt. Voor third-party apps waarvan we zeker weten dat ze de locatie niet over internet versturen, blijkt dan weer dat verschillende personen onterecht denken dat hun locatiegegevens toch verstuurd worden.

Terwijl de meerderheid aangaf niet wakker te liggen van hun privacy bij routeplanning applicaties, blijkt toch dat het 85 en 77 percent van de respondenten zou storen moesten respectievelijk hun locatie en reisplannen over internet verstuurd worden.

Overstappen naar een andere applicatie is voor velen echter een brug te ver: slechts 35 en 37 percent van de respondenten zou overstappen naar een applicatie die respectievelijk hun reisplannen en locatie niet over internet verstuurd. Een ongeveer even groot aandeel geeft aan dat ze dit misschien zouden doen, afhankelijk van wat de alternatieven zijn. Deze aantallen dienen we ook onmiddellijk te nuanceren: ondanks recente privacyschandalen, blijft het overgrote deel van de smartphonegebruikers messaging apps als Facebook Messenger en Whatsapp gebruiken, ondanks de beschikbaarheid van veiligere en privacyvriendelijkere applicaties. Overstappen en wennen aan een nieuwe applicatie kost moeite en tijd, wat veel gebruikers er niet voor over hebben.

Tot slot werden gebruikers bevraagd naar *wat ze vinden van een applicatie op basis van Linked Connections*. Twee personen gaven aan de uitleg niet volledig te begrijpen, en zijn van deze analyse uitgesloten. Respondenten werden gevraagd om de vier voornaamste voordelen van Linked Connections voor gebruikers van meest naar minst belangrijk te ordenen. Ondanks dat de gebruikers geïnformeerd werden dat Linked Connections volledige privacy biedt, en dat ongeveer een derde aan gaf over te stappen naar een privacyvriendelijke applicatie, komt privacy slechts bij 15% op de eerste plaats, en bij meer dan de helft zelfs op de vierde plaats. Snelheid blijft koploper, gevolgd door offline zoeken, aangepaste routes en tot slot privacy.

Dat aanpassen van routeplanning en privacy slechts op de vierde plaats komen wilt niet zeggen dat men dit niet belangrijk vindt. Tijdens user-tests gaven testers reeds aan deze rangschikkingen moeilijk te vinden, en wanneer gepolst wordt naar de interesse in afzonderlijke aspecten, blijkt dat mensen vooral het aanpassen van routeplanning en offline zoeken enorm interessant vinden, gevolgd door snelheid en privacy. Bij het aanpassen van routeplanning willen reizigers vooral kunnen zoeken naar routes met een kortere overstaptijd, drukke treinen mijden, en routes plannen met wat meer tijd om over te stappen. Privacy, wat op de laatste plaats staat, blijft interessant voor 83% van de respondenten. Hieruit kunnen we besluiten dat Linked Connections enorm veel potentieel heeft voor mobiele applicaties. We zullen deze resultaten nog verder bespreken in hoofdstuk 5.

4.5.2 Conclusies op basis van de enquête

We zullen nu een antwoord formuleren op de in hoofdstuk 3 geformuleerde vragen.

- Biedt offline informatie een meerwaarde voor gebruikers?
  Ja, gebruikers hebben grote interesse in offline opzoekingen, voornamelijk door een slechte mobiele internetverbinding tijdens het reizen, en in mindere mate omdat ze vrezen te veel data te verbruiken of gewoonweg niet over mobiel internet beschikken.
- Hecht de gebruiker belang aan privacy bij het gebruik van routeplanning apps? Zo ja, in welke mate?
  De gebruiker hecht in beperkte mate belang aan zijn of haar privacy bij gebruik van routeplanning apps. De helft zegt zich hier geen zorgen over te maken, maar slechts een minderheid van de gebruikers weet welke data over internet verzonden worden. Een derde van de gebruikers zou overstappen naar apps die privacyvriendelijker zijn.
- Heeft de gebruiker schrik om te veel mobiele data te verbruiken?
  Ongeveer de helft van de gebruikers heeft schrik om te veel mobiele data te gebruiken, al maakt slechts een derde van de gebruikers zich hier ernstig zorgen om.
- Hecht de gebruiker belang aan dataverbruik bij het gebruik van routeplanning apps?
  Één op zes gebruikers geeft aan soms geen informatie met een applicatie op te zoeken uit vrees te veel data te verbruiken. Wanneer gebruikers echter gevraagd wordt om een aantal aspecten van een routeplanning applicatie van belangrijk naar onbelangrijk te rangschikken, eindigt dataverbruik op de laatste plaats. Dataverbruik is voor de gebruiker dus van ondergeschikt belang aan de functionaliteit.
- Is de gebruiker tevreden met de snelheid van zijn huidige routeplanning app?
  Thuis zijn zeven op tien gebruikers tevreden met de snelheid van routeplanning applicaties. Onderweg daalt dit tot een derde, hoogstwaarschijnlijk door slechte netwerkverbindingen.

- Wat is voor een gebruiker belangrijk in routeplanning apps?
  Gebruikers vinden vooral het snel laden van resultaten belangrijk. Na snelheid volgen offline zoekopdrachten, waarna privacy, batterijgebruik en dataverbruik ongeveer even belangrijk zijn.

- Is de gebruiker geïnteresseerd in routeplanning op maat? Zo ja, welke aspecten spreken hem dan aan?
  De gebruiker is zeer geïnteresseerd in routeplanning op maat, waarbij vooral het aanpassen van de overstaptijd in stations belangrijk is, en ook het mijden van drukke treinen als zeer interessant beschouwd wordt.

- Is de gebruiker geïnteresseerd in offline opzoekingen?
  Zoals eerder vermeld vormen offline opzoekingen een meerwaarde voor gebruikers. Wanneer expliciet bevraagd, blijkt dan ook dat veel gebruikers hier in grote mate in geïnteresseerd zijn.

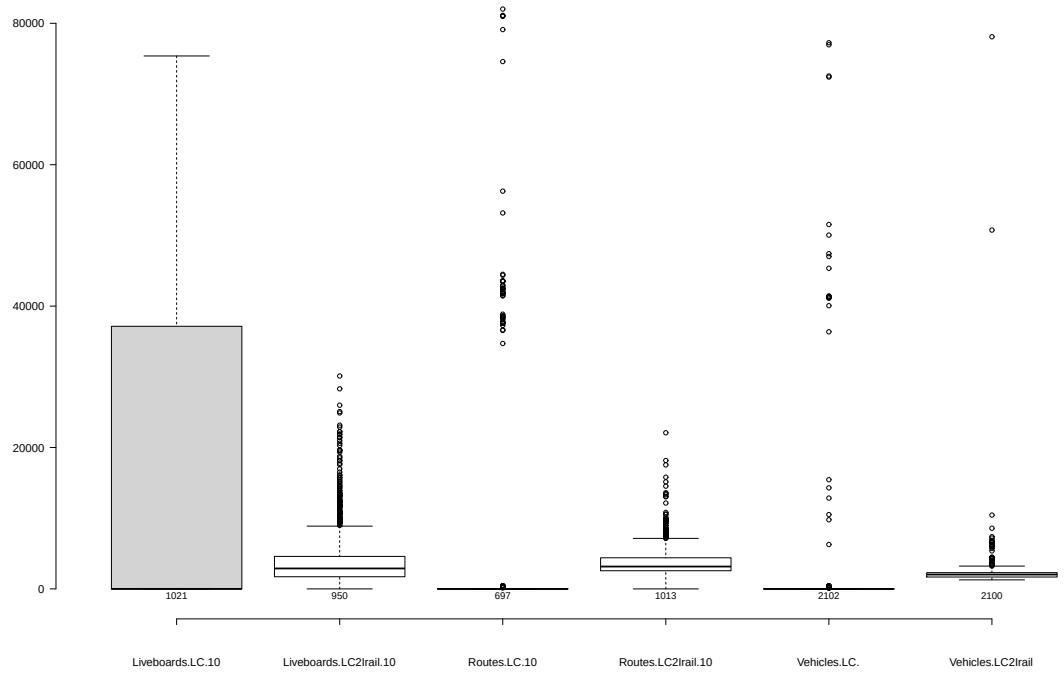
- Is de gebruiker geïnteresseerd in de mogelijke snelheid die Linked Connections biedt?
  Ondanks dat veel gebruikers al tevreden zijn met de huidige snelheid van routeplanning applicaties, blijft het overgrote deel geïnteresseerd in het verder versnellen van deze applicaties.

- Is de gebruiker geïnteresseerd in de volledige privacy die Linked Connections biedt?
  Ondanks dat privacy steevast onderaan de lijst met prioriteiten van gebruikers staat, wil dit niet zeggen dat gebruikers hierin niet geïnteresseerd zijn. Meer dan acht op tien gebruikers vindt dit interessant.

4.6 Dataverbruik

Zoals eerder aangehaald is op mobiele toestellen niet enkel de snelheid, maar ook andere zaken zoals batterijverbruik en connectiviteit van belang. We gaan nu dieper in op het dataverbruik van de applicatie.

Wanneer we dit dataverbruik onderzoeken, zichtbaar in figuur 4.25, zien we dat Linked Connections zoals te verwachten meer data gebruikt in de slechtste gevallen. Wat echter ook opvalt, is dat de mediaan steevast nul is. Dit is een rechtstreeks gevolg van de enorme cachebaarheid van Linked Connections. Linked Connections heeft ook last van grote uitlopers wanneer informatie wordt opgezocht met betrekking tot stations met weinig stops. In deze gevallen is het mogelijk dat Linked Connections alle vertrekken voor de komende paar dagen op moet halen, terwijl LC2Irail niet meer data dan anders moet versturen. LC2Irail zal immers steeds minstens één resultaat teruggeven, waardoor in het slechtste geval 10 verzoeken nodig zijn. Linked Connections kent



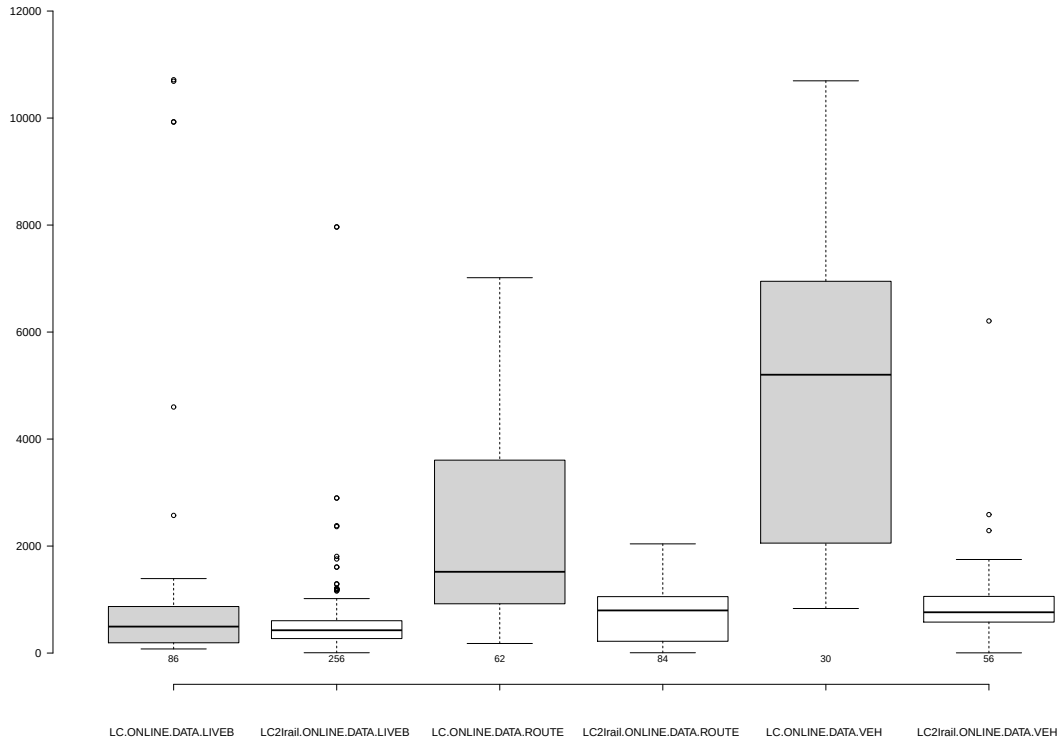
Figuur 4.25: Dataverbruik per type opzoeking en techniek

geen harde bovengrens voor het aantal verzoeken, al is het zinloos om langer dan 15 seconden te zoeken [25].

In het geval van Linked Connections zien we duidelijke verschillen tussen het opzoeken van liveboards en andere types data. Dit is te verklaren door het feit dat om een liveboard van een station te reconstrueren vaak slechts één of twee pagina's nodig zijn. Hierdoor is minder data nodig, maar wordt ook minder data gecachet. Opzoekingen voor verschillende tijdstippen hebben verschillende data nodig, terwijl deze nog niet gecachet is, waardoor minder vaak de cache gebruikt kan worden. Bijgevolg hebben meer dan 25% van de opzoekingen nieuwe data nodig. Wanneer we echter kijken naar dataverbruik voor routes en voertuigen, gebruiken deze verzoeken erg veel data. Hierdoor is er echt vaker een overlap tussen opzoekingen. Als gevolg wordt hier bij de enkele verzoeken enorm veel data binnengehaald, waarna vrijwel alle verzoeken uit cache beantwoord kunnen worden. We zien dat ook het derde kwartiel nul bedraagt, wat wilt zeggen dat meer dan 75% van de verzoeken uit cache geladen kan worden. Dit laden zien we in de uitlopers, die tot 2,4 megabyte kunnen oplopen.

Door data vooraf te laden wanneer de gebruiker verbonden is via Wi-Fi, zouden we ook deze uitlopers kunnen vermijden. In dit geval zou eerst een antwoord berekend kunnen worden op basis

van offline (verouderde) gegevens, waarna alle gebruikte pagina's opnieuw opgehaald worden om het actuele antwoord op te bouwen.



Figuur 4.26: Dataverbruik in werkelijke omstandigheden per type opzoeking voor Linked Connections

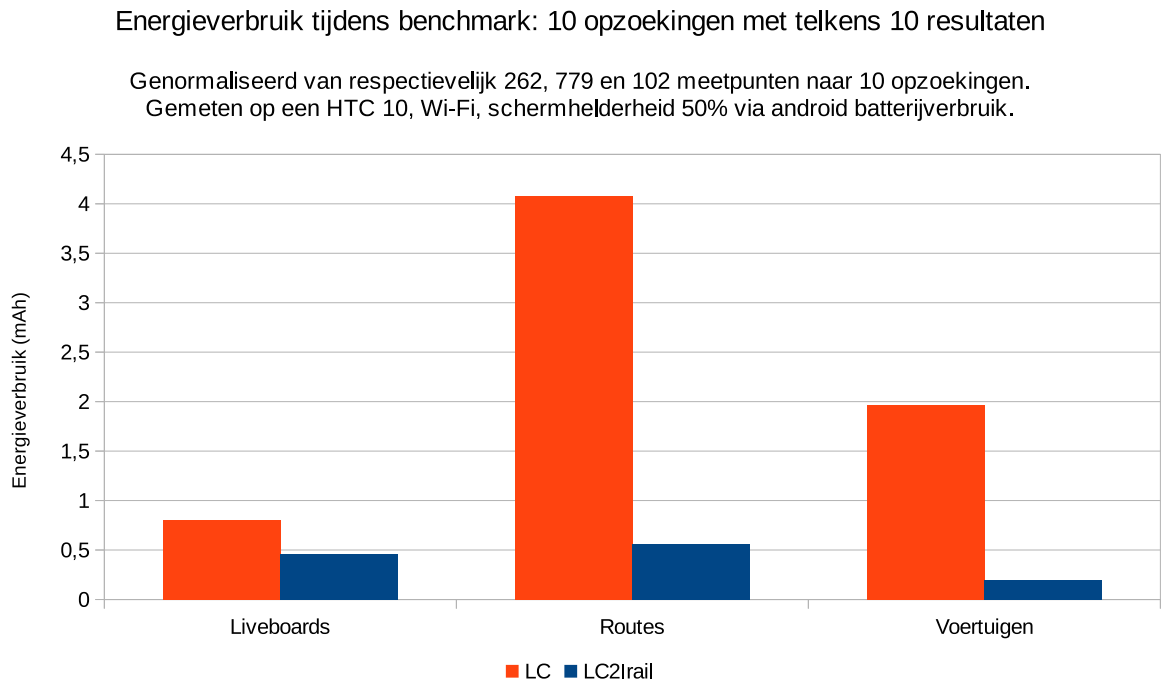
Bij deze cijfers dienen we een belangrijke kanttekening te maken. Deze gegevens tonen aan dat Linked Connections enorm cachbaar is, en enorm kan variëren in dataverbruik. Dit in tegenstelling tot een RPC API, die bijna altijd internet nodig heeft, met uitzondering van enkele populaire opzoekingen, die tijdens spitsuur uit cache geladen konden worden. Echter zijn deze gebaseerd op opzoekingen uitgevoerd op een server door verschillende gebruikers, en is de cache extreem afhankelijk van vorige opzoekingen. Het exacte dataverbruik wordt met andere woorden bepaald door het gebruikspatroon van de applicatie, en door eventueel gebruik van technieken zoals prefetching. Ter illustratie is in figuur 4.26 het werkelijke dataverbruik tijdens online opzoekingen weergegeven (online opzoekingen zijn opzoekingen waarbij internet beschikbaar was, en de applicatie dus vrije keus had tussen cache en online data). In deze figuur zien we een veel realistischer beeld, waarbij dat er, ondanks dat sommige opzoekingen volledig uit cache beantwoord kunnen worden, meestal toch data opgehaald wordt. Ook bij deze grafiek plaatsen we kanttekeningen. Zo zijn de metingen voor liveboards en routes per incrementeel resultaat

(er zijn dus mogelijk meerdere dergelijke opzoekingen nodig) en is het gebruik van de applicatie alsnog kunstmatig. Gebruikers kunnen in werkelijkheid mogelijk trager opzoeken, waardoor minder verzoeken uit cache beantwoord zouden kunnen worden.

4.7 Batterijverbruik

Naast dataverbruik is ook energieverbruik belangrijk bij draagbare toestellen. We gaan nu dieper in op het batterijverbruik van de applicatie. Dit is moeilijk om exact te meten, gezien alle omstandigheden exact dezelfde moeten zijn. Automatische testen op UI vereisen een USB-verbinding, waarbij stroom via USB geleverd wordt in plaats van via de batterij.

Om toch consistent verschillende applicaties te kunnen testen, maken we een variant op de testapplicatie, waarin we knoppen plaatsen om een korte benchmark uit te voeren. Hierdoor kunnen we dezelfde test voor elk soort opzoeking uitvoeren, waarna we uit Android energiebeheer het verbruik aflezen en het toestel weer tot 100% opladen alvorens de volgende test uit te voeren. Om aan testdata te komen kiezen we voor elk endpoint 5% van de opzoekingen uit de iRail-logs. De resultaten van deze tests zijn te zien in figuur 4.27.



Figuur 4.27: Gemiddeld energieverbruik per type opzoeking en techniek. Telkens worden tien resultaten geladen, voldoende om het scherm van de gebruiker te vullens.

Onmiddellijk vallen enkele verschillen tussen beide implementaties op. Zo zien we dat Linked

Connections steeds een veelvoud aan energie verbruikt. Dit varieert tussen dubbel zoveel voor Liveboards, tot acht maal zoveel voor routes. We zien hier een duidelijk verband tussen energieverbruik, de hoeveelheid gedownloade data en de complexiteit van de toegepaste algoritmes. Liveboards, die weinig data en weinig processing vereisen, verbruiken het minst energie. Voertuigen, die meer data vereisen maar nog steeds een relatief eenvoudig algoritme hebben verbruiken ongeveer drie keer meer energie. Routes vereisen iets minder data dan voertuigen, maar hebben een complexer algoritme. De processortijd blijkt een grote impact te hebben op de batterij, met een verdubbeling ten opzichte van het energieverbruik bij voertuigen tot gevolg.

Een belangrijke nuance bij deze resultaten is dat zelfs in het slechtste geval, het laden van routes via Linked Connections, slechts 4mAh verbruikt wordt. Dit is zeer weinig, en zal zelfs op smartphones met een zeer kleine batterij (2.000mAh) slechts 0,2% batterij verbruiken. Op toestellen met een gemiddelde batterij (3.200mAh) zakt dit verder tot 0,125%. Linked Connections verbruikt wel degelijk meer dan LC2Irail, maar het effect op de gebruiker blijft steeds beperkt.

4.8 Beperkingen van dit onderzoek

Het grootste deel van deze masterproef is gebaseerd op user-testing en benchmarks van de ontwikkelde applicatie. Deze tests zijn echter aan enkele beperkingen onderhevig. Deze beperkingen hebben geen grote impact op het onderzoek en hebben geen invloed op de uiteindelijke conclusies, maar kunnen de volledigheid of precisie van het onderzoek lichtjes beïnvloeden.

4.8.1 Kleine steekproef voor user-testing

Zoals eerder vermeld ontbreekt op het moment van schrijven nog cruciale informatie in de Linked Connections server implementatie voor de NMBS, zoals of een voertuig al dan niet afgeschaft is, en op welk perron een voertuig aankomt. Hierdoor moesten we terugvallen op user-testing onder begeleiding, om gebruikers aan te sporen hun gebruikelijke opzoekingen te doen en te polsen naar hun ervaringen. Dit neemt relatief veel tijd in beslag, waardoor weinig mensen én zin, én tijd hebben. Voorts neemt deze methode van testen ook veel tijd in beslag voor de onderzoeker.

De groep testgebruikers is wel gevarieerd, zowel in persoonlijke eigenschappen zoals leeftijd, als in reisgewoontes per trein. Wanneer de gehele testgroep duidelijk de voorkeur geeft aan een bepaalde variant, kunnen we deze keuze veralgemenen naar de gehele populatie. Wanneer er echter geen grote meerderheid voor eenzelfde variant kiest, moeten we voorzichtig zijn met conclusies.

4.8.2 Beperkt aantal unieke toestellen getest

Uit de voorgaande secties blijkt dat het gebruikte toestel van groot belang is voor de prestaties van de lokale Linked Connections implementatie. Tijdens het user-testen werd gebruik gemaakt van twaalf verschillende smartphones. Dit aantal is relatief beperkt in vergelijking met het aanbod op de huidige smartphonemarkt. Eventuele verder onderzoek zal de prestaties van Linked Connections op verschillende toestellen moeten vastleggen.

4.8.3 Processorverbruik niet meetbaar

De Android CPU-Profiler beïnvloedt de prestaties van de applicatie zodanig dat het onmogelijk is om een correct beeld te krijgen van het processorverbruik. Er kan een beeld gevormd worden welke onderdelen van de applicatie het meest processortijd vragen, maar exacte tijdsmetingen zijn niet mogelijk. Deze problemen worden ook door andere Android ontwikkelaars op internet beschreven[1]. Deze problemen treden op door de nieuwe Android CPU profiler, die zelf teveel processortijd op het apparaat vereist.

4.8.4 Prestaties zijn sterk afhankelijk van implementatiedetails

Zoals blijkt uit grafieken is de performantie van de lokale Linked Connections implementatie sterk afhankelijk van details in de implementatie - Het is dus niet enkel belangrijk om de algoritmes te optimalizeren, maar ook om rekening te houden met processen zoals Garbage Collection. Dit werd pas in een gevorderd stadium van de proef vastgesteld. Het is mogelijk dat de resultaten in dit onderzoek nog verder verbeterd kunnen worden door dezelfde algoritmes efficiënter te implementeren.

4.8.5 Kleine afwijkingen tussen opzoekingen LC en LC2Irail

Hoewel zowel Linked Connections en LC2Irail betrouwbaar werken in de testapplicatie, zijn er nog een beperkt aantal gevallen waarin het automatisch laden van incrementele resultaten niet werkt. Dit is onder andere mogelijk bij stations waar voor langer perioden geen voertuig stopt, of routes waarvoor slechts enkele resultaten per dag beschikbaar zijn. Om te zorgen dat ook voor deze opzoekingen incrementele resultaten feilloos laden zijn verdere verfijningen nodig aan de stopvoorwaarden en implementatie. Zo moet voorkomen worden dat er bij opzoekingen die data van meerdere dagen vereisen te veel callbacks gebruikt worden, een neveneffect dat tijdens de ontwikkeling nog niet bekend was. Aangezien enkel succesvolle opzoekingen meegenomen worden

[1] `https://stackoverflow.com/questions/49555983/background-concurrent-copying-gc-freed`

in testresultaten, is het hierdoor mogelijk dat er een klein verschil zit op het aantal meetpunten bij vergelijkende tests tussen LC en LC2Irail. Dit verschil treed vooral op bij het opzoeken van routes, en heeft geen merkbare invloed op resultaten door de grootte van de steekproef.

"It's difficult to imagine the power that you're going to have when so many different sorts of data are available."

~Tim Berners-Lee

5

Conclusie

We zullen nu de resultaten, besproken in hoofdstuk 4, interpreteren om zo een antwoord te vinden op de in sectie 1.3 geuite vragen.

Onderzoeksvraag Verbetert de user experience en user perceived perforance van een applicatie voor openbaar vervoer wanneer gebruik gemaakt wordt van Linked Connections in plaats van traditionele RPC API's?

Voor liveboards blijkt dat afhankelijk van het gebruikte toestel Linked Connections concurrentieel is op vlak van snelheid, zowel gemeten als ervaren. Dit wordt ook duidelijk gereflecteerd in de ervaren snelheid, die door een groot deel van de gebruikers als 'redelijk snel' of beter aangeduid wordt. Toch blijkt dat de ervaren snelheid iets onderdoet voor die van een RPC API.

Bij routes en voertuigen wordt de achterstand van Linked Connections tegenover RPC steeds groter. Gebruikers zijn het ook absoluut niet met elkaar eens over de ervaren snelheid, terwijl voor LC2Irail vrijwel alle gebruikers voor (ongeveer) dezelfde snelheid ervoeren.

Hieruit trekken we de conclusie dat Linked Connections enorm afhankelijk is van het gebruikte toestel, waarbij toestellen met meer processorkracht en/of werkgeheugen een duidelijk voordeel hebben ten opzichte van toestellen die over mindere specificaties beschikken.

Hoewel Linked Connections ten opzichte van LC2Irail traag kan overkomen, dienen we dit zeker te nuanceren. Zo is LC2Irail, de gebuikte referentie voor een RPC API, beduidend sneller dan Linked Connections. Echter blijkt dat de meeste gebruikers LC2Irail als sneller ervaren in vergelijking met de huidige beschikbare applicaties, en Linked Connections in veel gevallen als even snel als beschikbare applicaties. Wel zijn de prestaties van Linked Connections inconsistent en sterk afhankelijk van de gemaakte opzoeking, waardoor gebruikers sneller irritatie ervaren bij het gebruik van Linked Connections dan bij het gebruik van LC2Irail.

De user-perceived performance van Linked Connections is algemeen gezien, voor de meerderheid van de gebruikers, gelijk of iets slechter dan de user-perceived performance van RPC API's.

Opvallend is ook dat de meerderheid van de gebruikers, ondanks aan te geven dat ze LC2Irail sneller ervaren, toch aangeeft liefst Linked Connections te gebruiken wanneer ook offline toegang meespeelt. Dit wil zeggen dat gebruikers Linked Connections snel genoeg ervaren om een algemene betere gebruikerservaring te bieden vergeleken met RPC API's.

De user experience van een routeplanning applicatie op basis van Linked Connections is beter dan de user-experience van een identieke applicatie die gebruikt maakt van een RPC API, voornamelijk door een hoge betrouwbaarheid onafhankelijk van de beschikbaarheid van (mobiel) internet.

We gaan even dieper in op de prestaties, en proberen op basis van alle voorgaande informatie de achterliggende oorzaak van de variërende prestaties te achterhalen. Testpersonen die achtereenvolgens een implementatie op basis van de *org.json* parser en de *LoganSquare* voorgeschoteld kregen, gaven allemaal aan dat de *LoganSquare* parser betere prestaties bood, zowel op budget- als high-end smartphones. Hierbij werd telkens gemeten hoe lang het duurde om een Linked Connections pagina van JSON om te vormen tot een object. Dit is het enige verschil tussen beide implementaties, maar toch blijkt hier dat het 50e percentiel voor de uitvoeringstijd van deze code verdubbelde bij gebruik van de *LoganSquare* parser: in plaats van 100 zijn nu 200 milliseconden nodig.

Dit is volledig tegenstrijdig aan de gebruikerservaringen, en is dan ook moeilijk te vatten. Echter is er een zeer belangrijke 'externe' invloed op de uitvoeringstijd, namelijk de *Java Virtual Machine (JVM)* die de applicatie uitvoert. Deze JVM pauzeert de applicatie voor *garbage collection* wanneer er te veel *garbage* is - objecten die ooit gebruikt werden, maar waar nu geen enkele verwijzing meer naar bestaat. Tijdens deze *garbage collection* worden alle ongebruikte objecten verwijderd om geheugen vrij te maken. Hierbij komt het voordeel van de *LoganSquare* parser naar boven: ondanks dat het parsen op zich langer duurt, wordt aanzienlijk minder *garbage* gecreëerd, en is het aanzienlijk minder vaak nodig om de applicatie te pauzeren voor *garbage collection*.

Hierbij komt ook nog dat toestellen die over minder processorkracht beschikken, ook vaak over minder geheugen beschikken. Terwijl fabrikanten vrije keuze krijgen hoeveel heap geheugen ze ter beschikking stellen aan applicaties, correleert het advies hiervoor sterk met het beschikbaar RAM geheugen en de grootte en resolutie van het scherm. Als er minder geheugen beschikbaar is voor de applicatie, zal *garbage collection* vaker geheugen moeten vrijmaken.

Elke *garbage collection* zal door de beperktere processorkracht ook meer tijd vereisen, waardoor de applicatie niet alleen meer, maar ook langer gepauzeerd wordt. Hierdoor weegt Linked Connections extra zwaar door op trage toestellen: niet alleen kosten de algoritmes meer tijd, maar ook parsen van JSON kost meer tijd. Tragere modems kunnen er verder nog voor zorgen dat ook het netwerkverkeer trager gaat. Al deze factoren maken dat Linked Connections enorm afhankelijk is van het gebruikte toestel en de gebruikte programmeertaal, terwijl een RPC API zoals LC2Irail slechts weinig data over het netwerk verzendt, een klein antwoord heeft wat niet tot *garbage collection* leidt, en geen verdere algoritmes of verwerking vereist aan de client side.

Het zou onterecht zijn om Linked Connections definitief als 'slechter' te bestempelen. Wel kunnen we zeggen dat er zeer veel aandacht aan de exacte implementatie besteed moet worden, waarbij ontwikkelaars diepgaande kennis over hun omgeving moeten beschikken. Een ontwikkelaar die geen kennis heeft van de principes van *garbage collection*, zal sneller problemen ervaren bij de performantie van Linked Connections dan bij het implementeren van een RPC API.

Hypothese 1 Gebruikers beschikken niet over een kwalitatieve internetverbinding tijdens het reizen, en ervaren de mogelijkheid voor offline zoekopdrachten als een meerwaarde. Gebruikers hebben grote interesse in offline opzoekingen, voornamelijk door een slechte mobiele internetverbinding tijdens het reizen, en in mindere mate omdat ze vrezen te veel data te verbruiken of gewoonweg niet over mobiel internet beschikken.

Hieruit volgt dat Hypothese 1 correct is, en de gebruiker dit inderdaad als meerwaarde ervaart, op basis van een populatie bestaand uit 81 treinreizigers.

Hypothese 2 De gebruiker ervaart de mogelijkheid om voorkeuren voor routes in te stellen (overstaptijd, toegankelijkheid, ...) als een meerwaarde. De gebruiker is zeer geïnteresseerd in routeplanning op maat, waarbij vooral het aanpassen van de overstaptijd in stations belangrijk is, en ook het mijden van drukke treinen als zeer interessant beschouwd wordt.

Hieruit volgt dat Hypothese 2 correct is, en de gebruiker dit inderdaad als meerwaarde beschouwt.

Hypothese 3 De gebruiker heeft angst om te veel mobiele data te verbruiken, maar let hier niet op bij het gebruik van routeplanning-apps. 53% van de gebruikers is nog voorzichtig met zijn of haar databundel. Slechts 14% heeft 'redelijk veel' schrik om te veel data te verbruiken, 19% heeft 'veel' of 'enorm veel' schrik om te veel data te verbruiken. Gebruikers zijn dus zeker nog voorzichtig met hun databundel, al maakt 47% zich hier geen zorgen over. Wanneer we kijken naar datagebruik bij routeplanning applicaties, blijkt dat één op zes gebruikers soms geen informatie met een applicatie op te zoeken uit vrees te veel data te verbruiken. Dit komt ongeveer overeen met de personen die aangaven nog veel schrik te hebben om te veel data te verbruiken.

Wanneer gebruikers echter gevraagd wordt om een aantal aspecten van een routeplanning applicatie van belangrijk naar onbelangrijk te rangschikken, eindigt dataverbruik op de laatste plaats. Dataverbruik is voor de gebruiker dus van ondergeschikt belang aan de functionaliteit.

De helft van de gebruikers heeft angst om te veel mobiele data te gebruiken, maar slechts één op zes gebruikers let ook op bij het gebruik van routeplanning applicaties op mobiel datagebruik. Hypothese 3 blijkt correct.

Hypothese 4 De gebruiker ervaart extra privacy bij het opzoeken van routes niet als een noemenswaardige meerwaarde Gebruikers maken zich niet echt zorgen om hun privacy bij het gebruik van routeplanning applicaties. Verder zijn ze ook zeer slecht op de hoogte welke data over internet verzonden worden. Privacy belandt ook steevast onderaan de lijst met prioriteiten voor gebruikers. Dit wil echter niet zeggen dat men hierin niet geïnteresseerd is. Wanneer expliciet naar interesse om meer privacy gepolst wordt, geeft meer dan acht op tien gebruikers aan dit interessant te vinden.

Gebruikers ervaren extra privacy als een meerwaarde, alhoewel deze niet noemenswaardig is, en niet noemenswaardig veel gebruikers zal overhalen over te stappen. Ook hypothese 4 blijkt correct.

5.1 Tot slot

In deze masterproef hebben we een client geschreven om informatie over openbaar vervoer per trein te raadplegen, gebruikmakend van Linked Connections. Hiervoor hebben we algoritmes uitgewerkt om op een efficiënte manier vertrekken en aankomsten in een station op te lijsten en het traject van een voertuig weer te geven. Ook hebben we routeplanning geïmplementeerd op basis van het Connection Scan Algoritme en geoptimaliseerd zodat niet de snelste, maar de beste route bepaald wordt, en zodat informatie over de volledige reis opgehaald kan worden.

Ook hebben we ondervonden dat implementatiedetails een groot verschil in prestaties kunnen veroorzaken, en de implementatie nog aangepast om gebruik te maken van een efficiëntere JSON parser. Tevens ontwikkelden we LC2Irail, een RPC API die dezelfde algoritmes en data gebruikt als de lokale Linked Connections client. LC2Irail werd gebruikt als referentie voor een RPC API.

Vervolgens hebben we in hoofdstuk 3 uiteengezet hoe we aan de hand van deze cliënt de performance, user-perceived performance en user experience bepalen. Vervolgens zijn volgens de hier omschreven methode user-tests afgenomen, is er een enquête uitgevoerd en zijn de prestaties gemeten bij automatische tests.

De resultaten van al deze tests zijn uitvoerig besproken in hoofdstukken 4 en 5. Hierbij blijkt dat LC2Irail beter presteert dan Linked Connections, alhoewel Linked Connections nog net de snelheid van de huidige beschikbare applicaties kan volgen. Linked Connections blijkt een inconsistente ervaring te geven, terwijl LC2Irail een consistente snelle ervaring kan bieden. Linked Connections is aantoonbaar trager, en ook gebruikers ervaren dit zo. Desondanks biedt offline opzoeken een grote meerwaarde, waardoor de meeste gebruikers toch lieven Linked Connections gebruiken dan LC2Irail.

Tot slot kwamen we tot de conclusie dat de in hoofdstuk 1 gestelde hypotheses correct zijn, en dat de gebruikerservaring beter is bij het gebruik van Linked Connections ondanks dat de user perceived performance lager ligt.

Bibliografie

[1] M. Boyd. (2014) How smart cities are using APIs: Public transport APIs. [Online]. Available: https://www.programmableweb.com/news/how-smart-cities-are-using-apis-public-transport-apis/2014/05/22

[2] O. K. International. (2018) What is open? Open Knowledge Foundation. [Online]. Available: https://okfn.org/opendata/

[3] B. J. Nelson, "Remote procedure call," Ph.D. dissertation, Pittsburgh, PA, USA, 1981, aAI8204168.

[4] 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 *Lecture Notes in Computer Science*, vol. 8796. Springer, 2014, pp. 180–196. [Online]. Available: http://linkeddatafragments.org/publications/iswc2014.pdf

[5] P. Colpaert, A. Llaves, R. Verborgh, O. Corcho, E. Mannens, and R. V. D. Walle, "Intermodal public transit routing using linked connections," vol. 1486, 2015. [Online]. Available: http://ceur-ws.org/Vol-1486/paper_28.pdf

[6] 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

[7] 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

[8] 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

[9] M. Ní Chonchúir and J. McCarthy, "The enchanting potential of technology: a dialogical case study of enchantment and the Internet," *Personal and Ubiquitous Computing*, vol. 12, no. 5, pp. 401–409, Jun 2008. [Online]. Available: https://doi.org/10.1007/s00779-007-0157-0

[10] B. Vanhaelewyn and L. D. Marez, "Imec.digimeter 2017." [Online]. Available: https://www.imec.be/digimeter

[11] S. Ickin, K. Wac, M. Fiedler, L. Janowski, J. H. Hong, and A. K. Dey, "Factors influencing quality of experience of commonly used mobile applications," *IEEE Communications Magazine*, vol. 50, no. 4, pp. 48–56, April 2012.

[12] P. van Ammelrooy. Onbeperkt smartphone-abbonement steeds meer in trek onder nederlanders. [Online]. Available: https://www.volkskrant.nl/economie/onbeperkt-smartphone-abonnement-steeds-meer-in-trek-onder-nederlanders~a4532349/

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

[14] 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 connections," vol. 1931, 2017, pp. 1–4. [Online]. Available: https://biblio.ugent.be/publication/8540883/file/8540885.pdf

[15] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Intriguingly simple and fast transit routing," in *Experimental Algorithms*, V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 43–54. [Online]. Available: https://pdfs.semanticscholar.org/a892/a54b02ce112e1302931231141a8b676b873b.pdf

[16] B. Strasser and D. Wagner, "Connection Scan Accelerated," in *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2014, pp. 125–137. [Online]. Available: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973198.12

[17] 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

[18] P. Colpaert. (2018) Linked Connections: What is it? [Online]. Available: https://linkedconnections.org/#what

[19] P. Moore and H. V. Pham, "On context and the open world assumption," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, March 2015, pp. 387–392.

[20] 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

[21] M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, "Timetable information: Models and algorithms," in *Algorithmic Methods for Railway Optimization*, F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. D. Zaroliagis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 67–90. [Online]. Available: https://www.ceid.upatras.gr/webpages/faculty/zaro/pub/jou/J23-TTI-Springer.pdf

[22] Y. Disser, M. Müller-Hannemann, and M. Schnee, "Multi-criteria shortest paths in time-dependent train networks," in *Experimental Algorithms*, C. C. McGeoch, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 347–361. [Online]. Available: https://pdfs.semanticscholar.org/b480/e71299ab9557cc7dd09908c840c6eb1cf9f0.pdf

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

[24] M. Müller-Hannemann and M. Schnee, "Paying less for train connections with MOTIS," in *OASIcs-OpenAccess Series in Informatics*, vol. 2. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[25] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: ACM, 1968, pp. 267–277. [Online]. Available: http://doi.acm.org/10.1145/1476589.1476628

Bijlagen

A

Vragen enquete

- Hoe vaak neem je de trein?
- In welk(e) verband(en) neem je de trein?
- Waar haal je (realtime) informatie met betrekking tot treinen vandaan?
- Hoe vaak ervaar je volgende gebeurtenissen wanneer je met de trein reist, en waar zoek je in deze gevallen informatie op?
  - Een probleemloze rit
  - Vertraging
  - Afgeschafte treinen
  - Spoorwijzigingen
  - Informatie in stations niet up-to-date
  - Informatie in app niet up-to-date
- Hoe tevreden ben je over informatiebronnen voor openbaar vervoer per trein?
- Rangschik deze bronnen voor informatie voor openbaar vervoer per trein naar hoe vaak je ze gebruikt, van meest naar minst gebruikt.
  - Website

  - App
  - Affiches of digitale borden in station
  - Loketten
  - Omgeroepen informatie

- Welk besturingssysteem gebruik je op je (meestgebruikte) smartphone
- Welke app gebruik je hoofdzakelijk?
- Waar gebruik je deze app?
- Hoe tevreden ben je over de volgende zaken wanneer je je applicatie gebruikt?
- Hoe tevreden ben je over het mobiele netwerk tijdens een treinreis?
- Heb je soms last van een zeer trage of afwezige netwerkverbinding wanneer je op de trein zit, waardoor webpagina's enorm traag of zelfs niet laden?
- Heb je schrik om meer mobiele data te verbruiken dan in je gsm abonnement of prepaid-bundel zit?
- Als je informatie over treinen wenst en deze niet opzoekt via een applicatie, wat is hiervoor dan de reden?
- Hoe belangrijk vind je onderstaande zaken in een app voor openbaar vervoer per trein? Rangschik van meest naar minst interessant.
  - Offline zoekopdrachten
  - Weinig data verbruiken
  - Snel resultaten laden
  - Mijn privacy beschermen
  - Weinig batterij verbruiken
- Hoe bezorgd ben je om je privacy bij het gebruik van je applicatie?
- Denk je dat je applicatie je locatie of reisplannen over internet verstuurt?
- Zou het je storen als je applicatie je locatie of reisplannen over internet verstuurt?
- Zou je overschakelen van je applicatie naar een andere app, als deze andere app je locatie of reisplannen niet over internet verstuurt?
- Hoe interessant vind je deze aspecten? Snelheid, privacy, offline gebruik, aanpasbare routeplanning.

- Stel dat je in een app de routeplanning ook kon aanpassen. Hoe interessant zou je het vinden om ook deze parameters in te kunnen stellen?
  - Drukke treinen mijden
  - Specifieke treinen mijden
  - Kortere overstappen gebruiken
  - Langere overstappen gebruiken
  - Enkel langs stations met lift, roltrap, ... plannen
- Rangschik de volgende aspecten van Linked Connections van meest naar minst interessant: snelheid, privacy, offline gebruik, aanpasbare routeplanning.
- Hoe oud ben je?
- Wat is je geslacht?

B

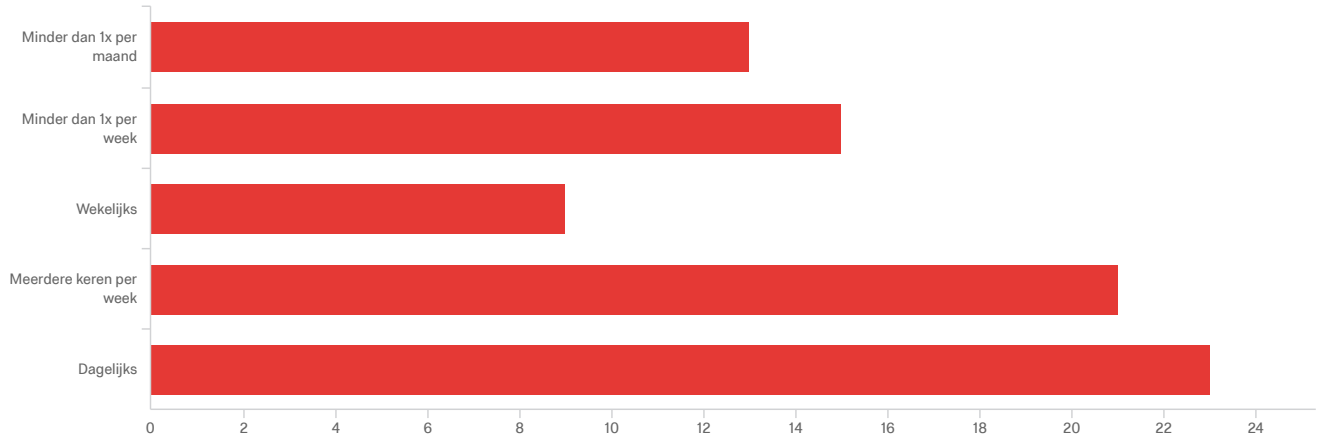
Resultaten enquete

Enquete W/O usertest

Thesis Linked Connections

May 30, 2018 2:21 PM CEST

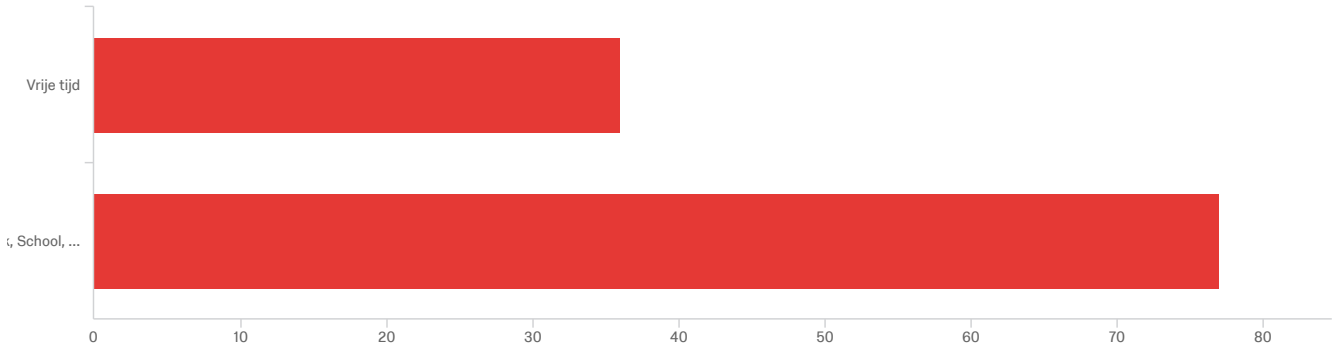
Q4 - Hoe vaak neem je de trein?



#	Field		Choice Count
1	Minder dan 1x per maand	16.05%	13
2	Minder dan 1x per week	18.52%	15
3	Wekelijks	11.11%	9
4	Meerdere keren per week	25.93%	21
5	Dagelijks	28.40%	23
			81

Showing Rows: 1 - 6 Of 6

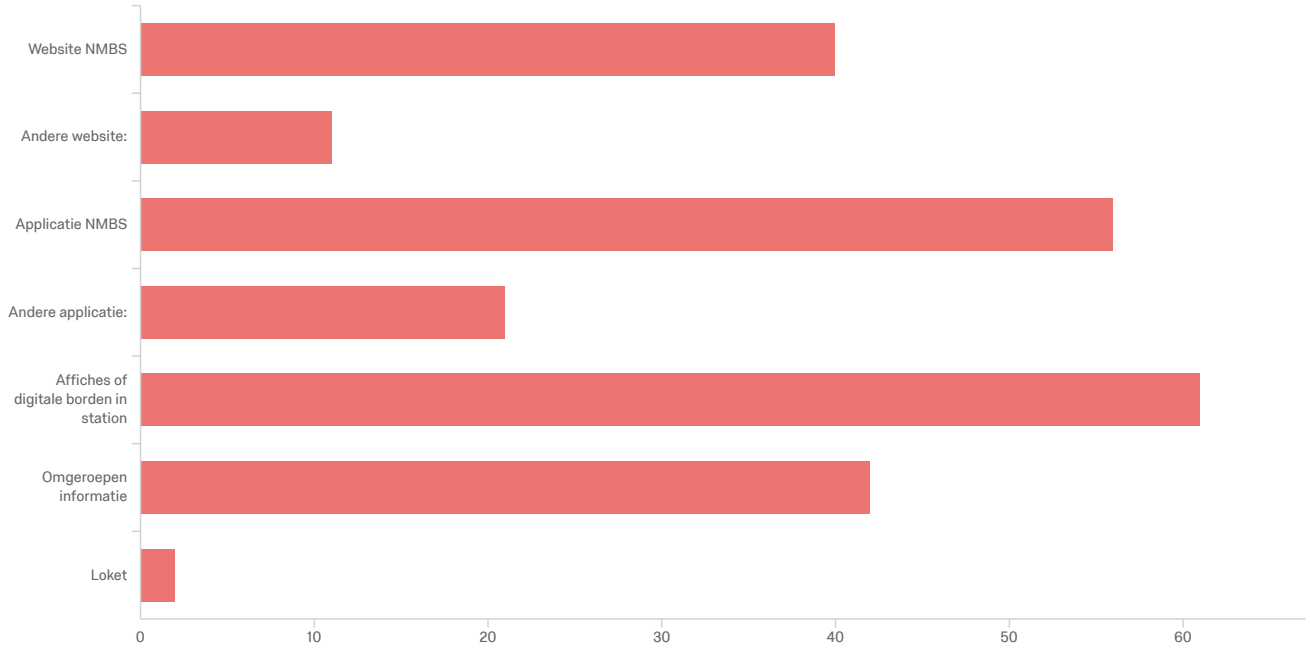
Q5 - In welk(e) verband(en) neem je de trein? Markeer alle toepasselijke antwoorden.



#	Field		Choice Count
1	Vrije tijd	31.86%	36
2	Werk, School, ...	68.14%	77
			113

Showing Rows: 1 - 3 Of 3

Q6 - Waar haal je (realtime) informatie met betrekking tot treinen vandaan? Markeer alle toepasselijke antwoorden.



#	Field	Choice Count	
1	Website NMBS	17.17%	40
2	Andere website:	4.72%	11
3	Applicatie NMBS	24.03%	56
4	Andere applicatie:	9.01%	21
5	Affiches of digitale borden in station	26.18%	61
6	Omgeroepen informatie	18.03%	42
7	Loket	0.86%	2
			233

Showing Rows: 1 - 8 Of 8

Q6_2_TEXT - Andere website:

Website (Anders)
de lijn
google maps

NS.nl
Http://www.IRail.be
Google
irail
www.eurrail.com
irail.be
irail.be
irail
irail.be

Showing Records: 1 - 11 Of 11

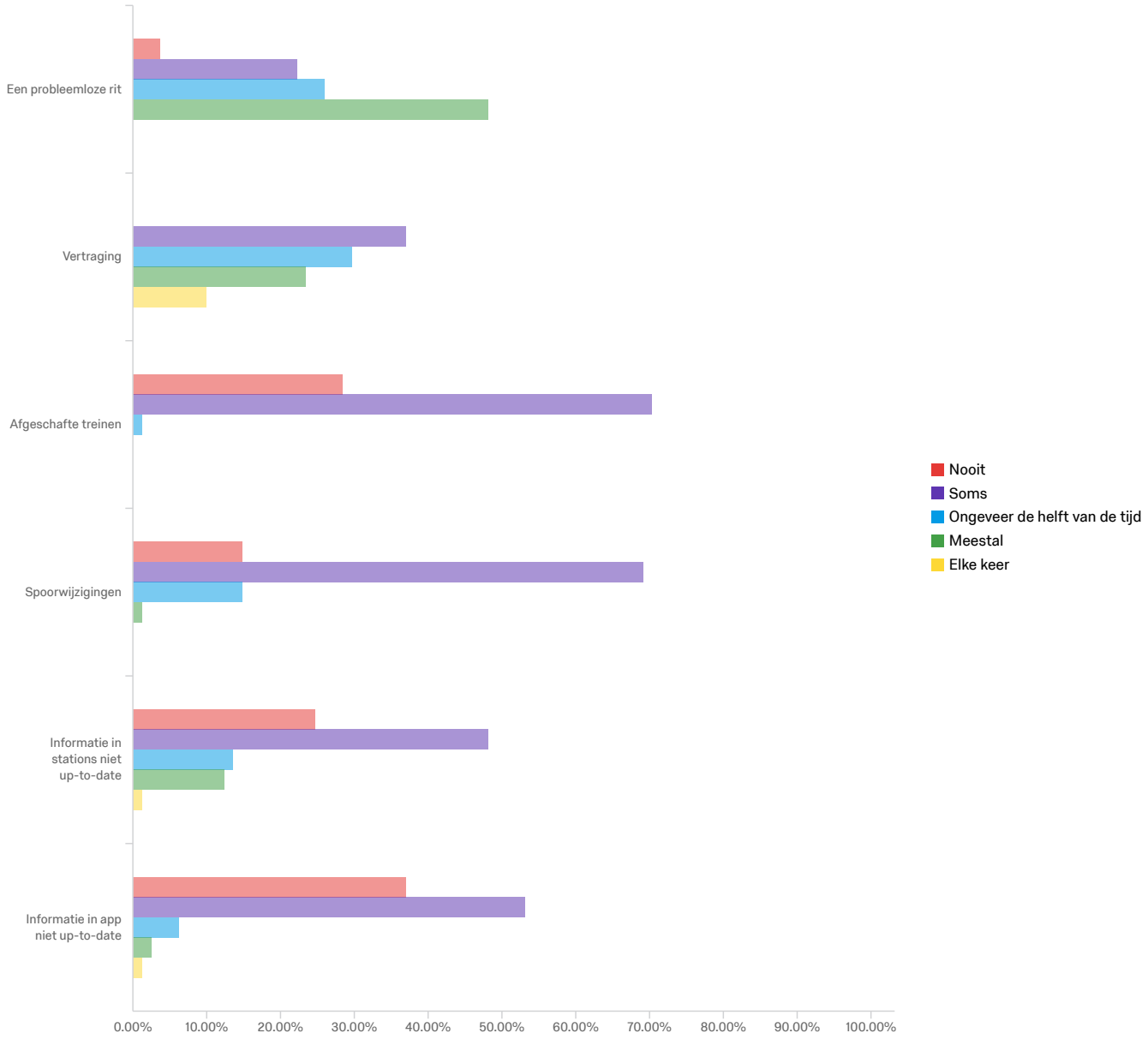
Q6_4_TEXT - Andere applicatie:

Applicatie (Anders)
twitter
Google maps
Citymapper
de lijn
De lijn
Railer
Railer
BeRail
9292
Google maps
Deutsche Bahn
Hyperrail
Railer

BeTrains (Android)
BeTrains, hyperrail, nexttrain
BeTrains
hyperrail
hyperrail
hyperrail
Hyperrail
Railer

Showing Records: 1 - 21 Of 21

Q7 - Hoe vaak ervaar je volgende gebeurtenissen wanneer je met de trein reist?

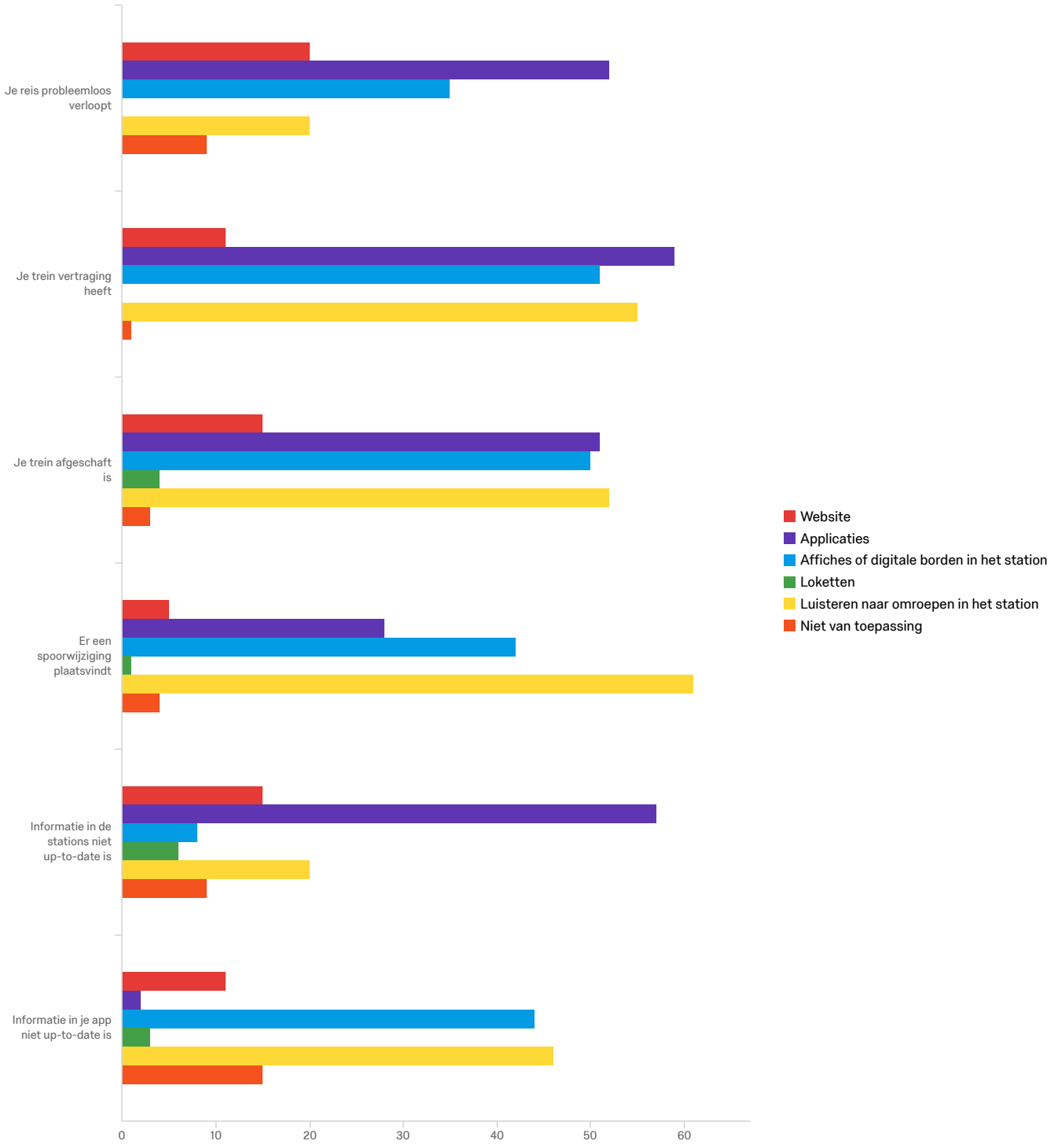


#	Field	Nooit	Soms	Ongeveer de helft van de tijd	Meestal	Elke keer	Total
1	Een probleemloze rit	3.70% 3	22.22% 18	25.93% 21	48.15% 39	0.00% 0	81
2	Vertraging	0.00% 0	37.04% 30	29.63% 24	23.46% 19	9.88% 8	81
3	Afgeschafte treinen	28.40% 23	70.37% 57	1.23% 1	0.00% 0	0.00% 0	81
4	Spoorwijzigingen	14.81% 12	69.14% 56	14.81% 12	1.23% 1	0.00% 0	81
5	Informatie in stations niet up-to-date	24.69% 20	48.15% 39	13.58% 11	12.35% 10	1.23% 1	81

6	Informatie in app niet up-to-date	37.04%	30	53.09%	43	6.17%	5	2.47%	2	1.23%	1	81

Showing Rows: 1 - 6 Of 6

Q8 - Waar zoek je informatie op als ...



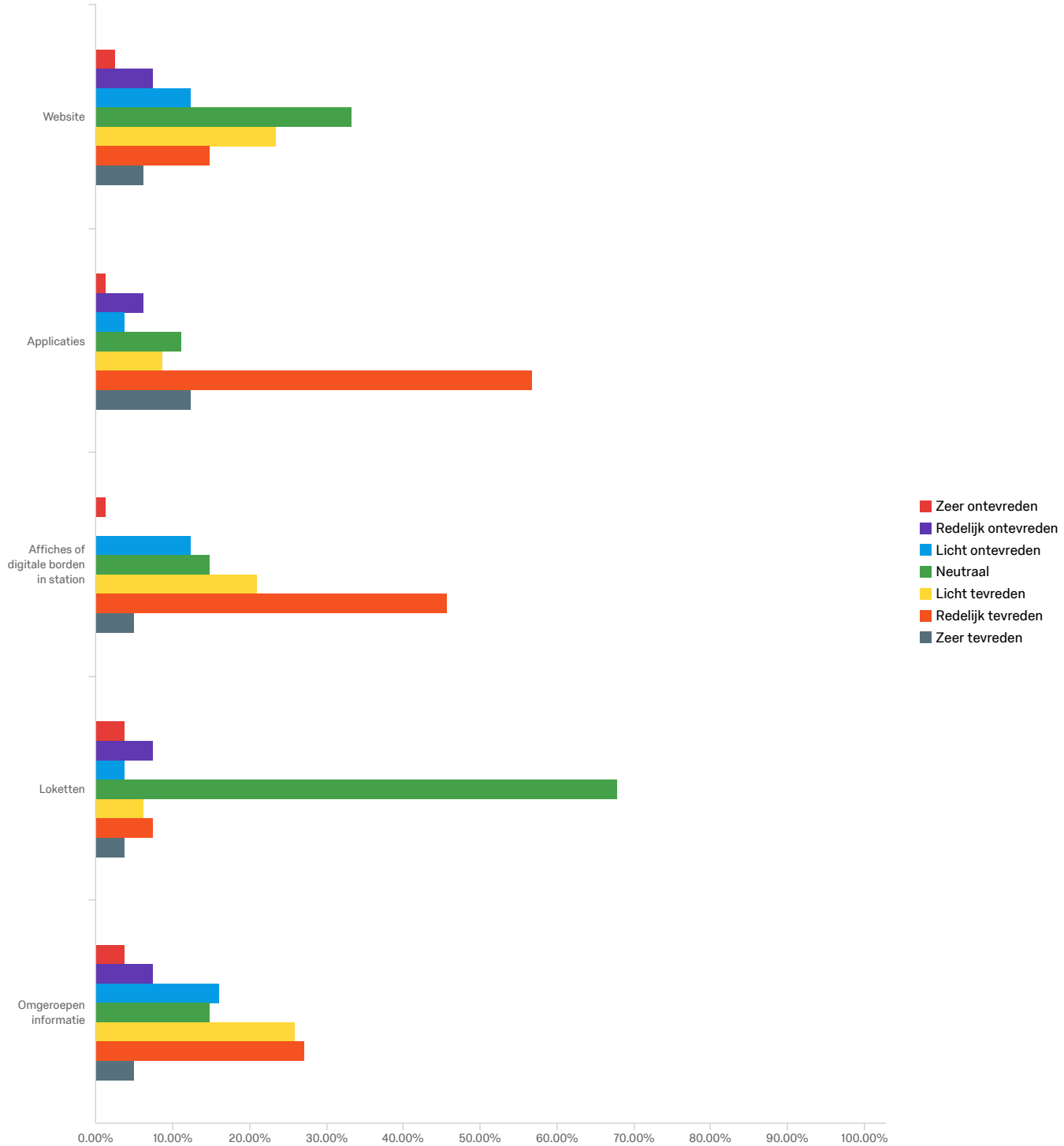
#	Field	Website	Applicaties	Affiches of digitale borden in het station	Loketten	Luisteren naar omroepen in het station	Niet van toepassing	Total
1	Je reis probleemloos verloopt	14.71% 20	38.24% 52	25.74% 35	0.00% 0	14.71% 20	6.62% 9	136

2	Je trein vertraging heeft	6.21%	11	33.33%	59	28.81%	51	0.00%	0	31.07%	55	0.56%	1	177
3	Je trein afgeschaft is	8.57%	15	29.14%	51	28.57%	50	2.29%	4	29.71%	52	1.71%	3	175
4	Er een spoorwijziging plaatsvindt	3.55%	5	19.86%	28	29.79%	42	0.71%	1	43.26%	61	2.84%	4	141
5	Informatie in de stations niet up-to-date is	13.04%	15	49.57%	57	6.96%	8	5.22%	6	17.39%	20	7.83%	9	115
6	Informatie in je app niet up-to-date is	9.09%	11	1.65%	2	36.36%	44	2.48%	3	38.02%	46	12.40%	15	121

Showing Rows: 1 - 6 Of 6

Q9 - Hoe tevreden ben je over deze informatiebronnen voor openbaar vervoer per trein?

Voor informatiebronnen die je niet gebruikt duidt je "neutraal" aan.

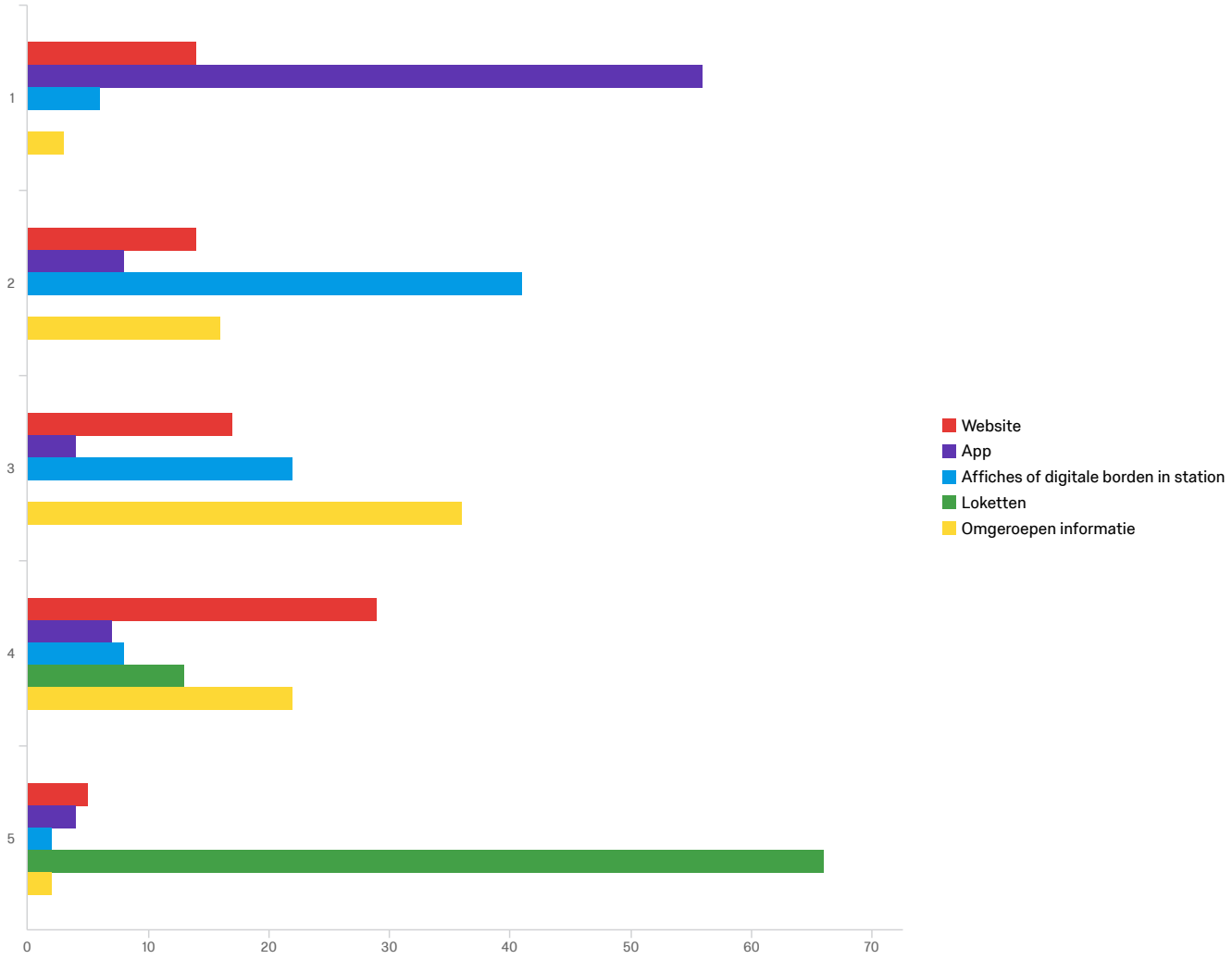


#	Field	Zeer ontevreden	Redelijk ontevreden	Licht ontevreden	Neutraal	Licht tevreden	Redelijk tevreden	Zeer tevreden

1	Website	2.47%	2	7.41%	6	12.35%	10	33.33%	27	23.46%	19	14.81%	12	6.17%	5
2	Applicaties	1.23%	1	6.17%	5	3.70%	3	11.11%	9	8.64%	7	56.79%	46	12.35%	10
3	Affiches of digitale borden in station	1.23%	1	0.00%	0	12.35%	10	14.81%	12	20.99%	17	45.68%	37	4.94%	4
4	Loketten	3.70%	3	7.41%	6	3.70%	3	67.90%	55	6.17%	5	7.41%	6	3.70%	3
5	Omgeroepen informatie	3.70%	3	7.41%	6	16.05%	13	14.81%	12	25.93%	21	27.16%	22	4.94%	4

Showing Rows: 1 - 5 Of 5

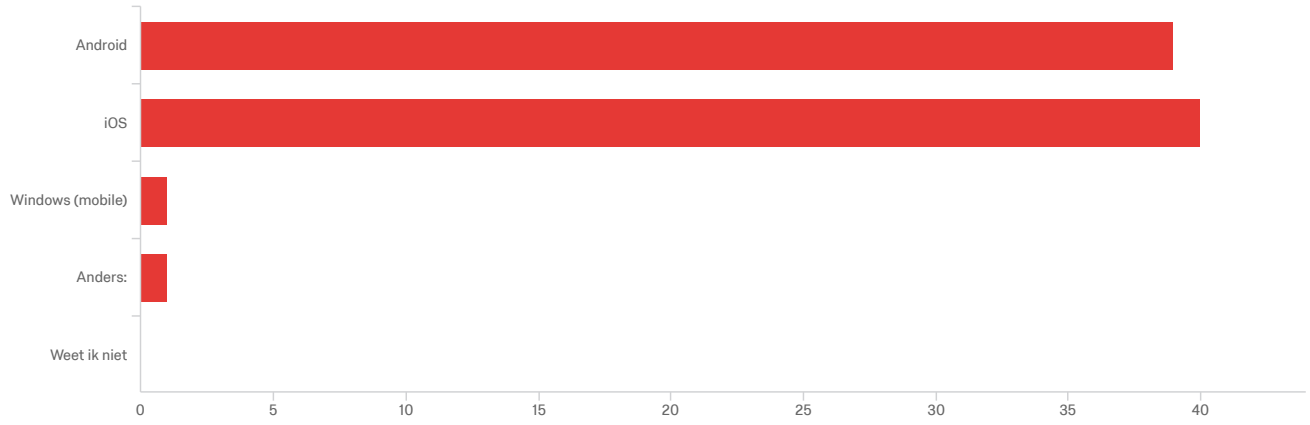
Q10 - Rangschik deze bronnen voor informatie voor openbaar vervoer per trein naar hoe vaak je ze gebruikt, van meest gebruikt naar minst gebruikt.



#	Field	1		2		3		4		5		Total
1	Website	17.72%	14	17.72%	14	21.52%	17	36.71%	29	6.33%	5	79
2	App	70.89%	56	10.13%	8	5.06%	4	8.86%	7	5.06%	4	79
3	Affiches of digitale borden in station	7.59%	6	51.90%	41	27.85%	22	10.13%	8	2.53%	2	79
4	Loketten	0.00%	0	0.00%	0	0.00%	0	16.46%	13	83.54%	66	79
5	Omgeroepen informatie	3.80%	3	20.25%	16	45.57%	36	27.85%	22	2.53%	2	79

Showing Rows: 1 - 5 Of 5

Q12 - Welk besturingssysteem gebruik je op je (meestgebruikte) smartphone



#	Field		Choice Count
1	Android	48.15%	39
2	iOS	49.38%	40
3	Windows (mobile)	1.23%	1
4	Anders:	1.23%	1
5	Weet ik niet	0.00%	0
			81

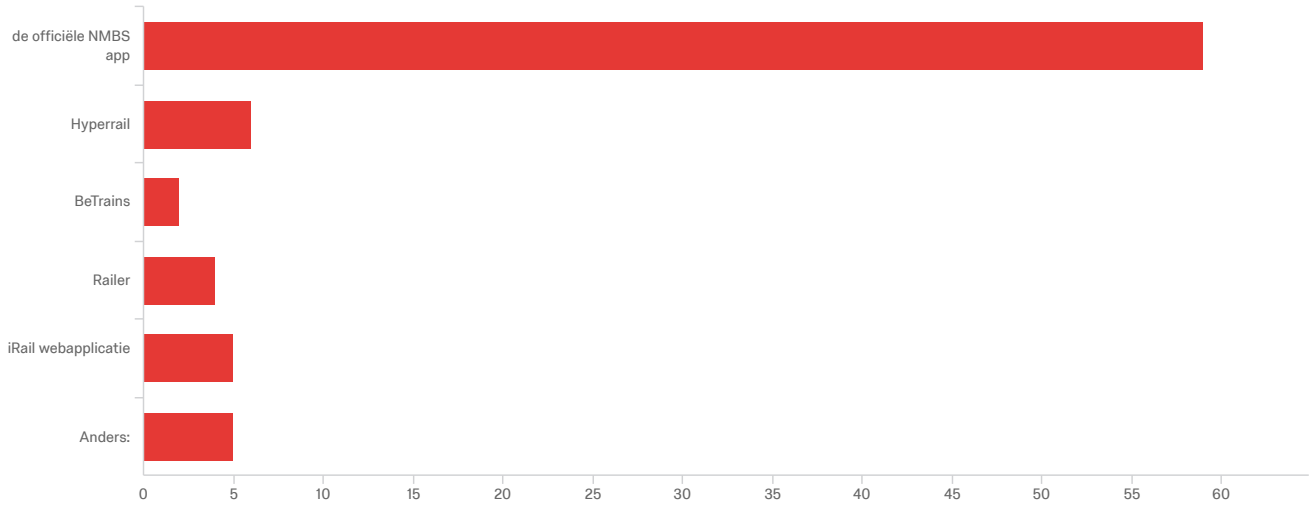
Showing Rows: 1 - 6 Of 6

Q12_4_TEXT - Anders:

Anders:
Sailfish OS

Showing Records: 1 - 1 Of 1

Q13 - Welke app gebruik je hoofdzakelijk?



#	Field		Choice Count
1	de officiële NMBS app	72.84%	59
2	Hyperrail	7.41%	6
3	BeTrains	2.47%	2
4	Railer	4.94%	4
5	iRail webapplicatie	6.17%	5
6	Anders:	6.17%	5
			81

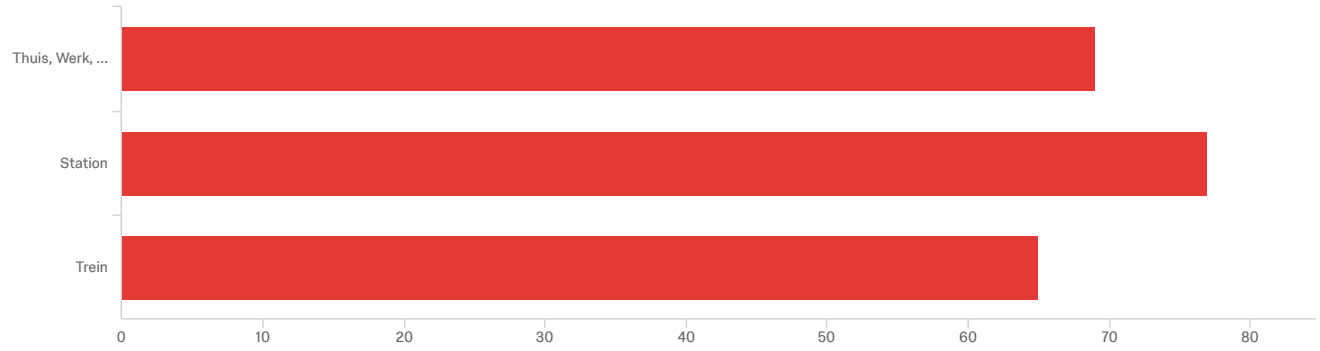
Showing Rows: 1 - 7 Of 7

Anders:

Anders:
citymapper
De Lijn
BeRail
9292.nl
Deutsche Bahn

Showing Records: 1 - 5 Of 5

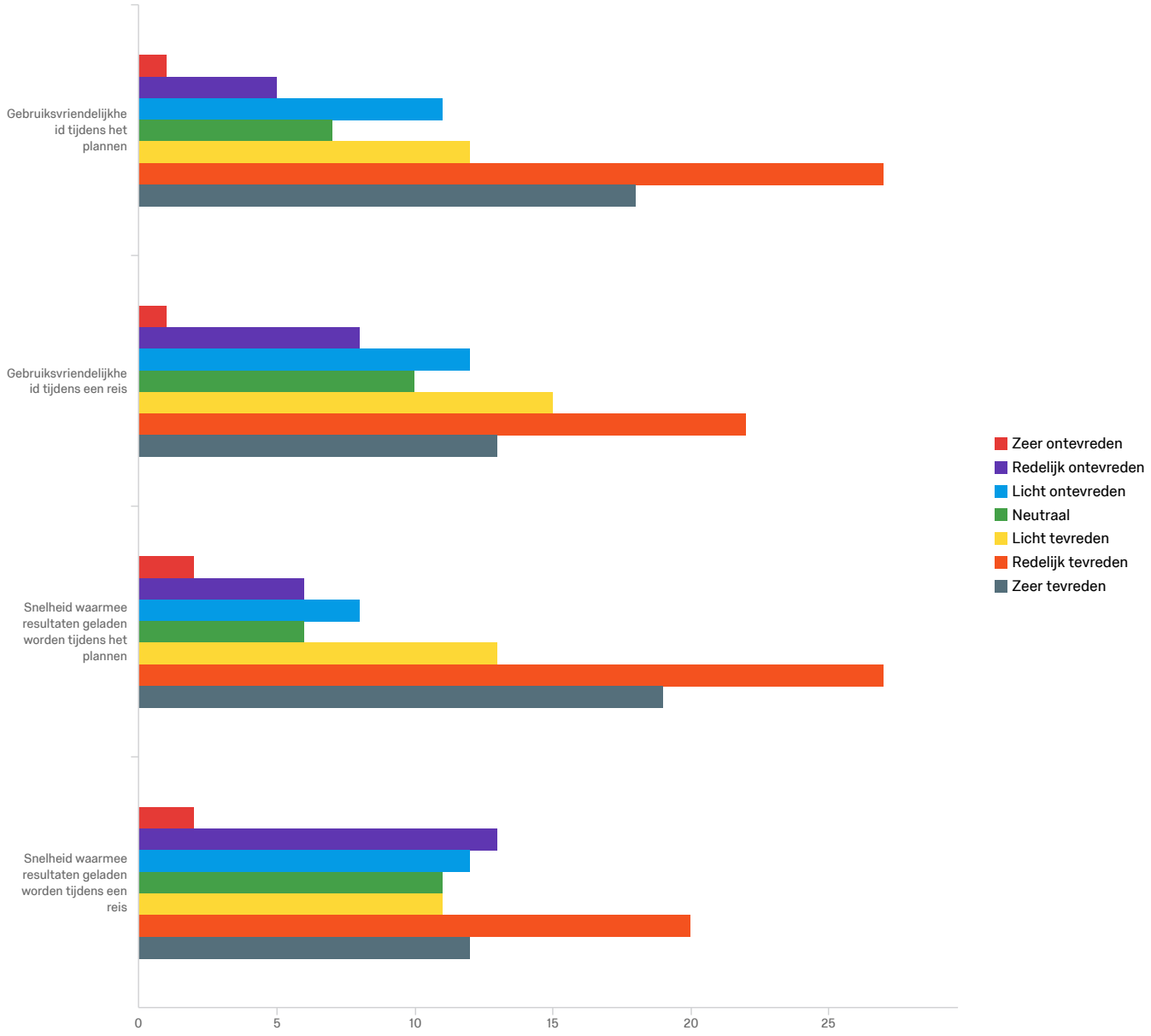
Q14 - Waar gebruik je deze app? Duid alle toepasselijke antwoorden aan.



#	Field		Choice Count
1	Thuis, Werk, ...	32.70%	69
2	Station	36.49%	77
3	Trein	30.81%	65
			211

Showing Rows: 1 - 4 Of 4

Q15 - Hoe tevreden ben je over de volgende zaken wanneer je [QID3-ChoiceGroup-SelectedChoicesTextEntry] gebruikt?

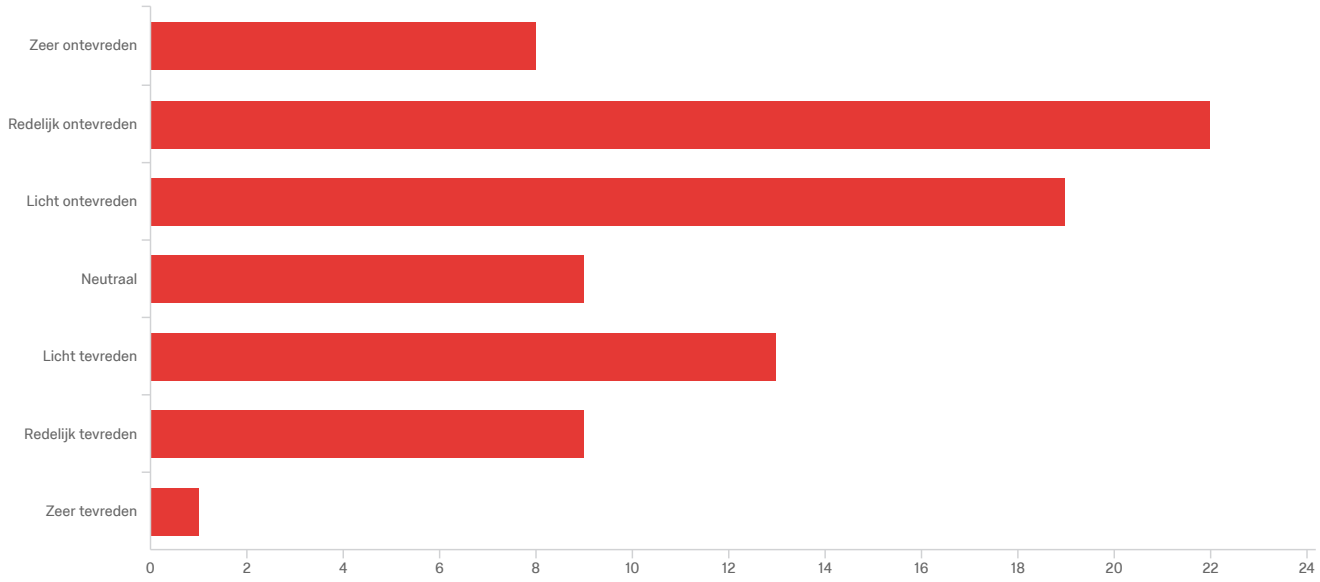


#	Field	Zeer ontevreden	Redelijk ontevreden	Licht ontevreden	Neutraal	Licht tevreden	Redelijk tevreden	Zeer tevreden
1	Gebruiksvriendelijkheid tijdens het plannen	1.23% 1	6.17% 5	13.58% 11	8.64% 7	14.81% 12	33.33% 27	22.22% 18
2	Gebruiksvriendelijkheid tijdens een reis	1.23% 1	9.88% 8	14.81% 12	12.35% 10	18.52% 15	27.16% 22	16.05% 13

3	Snelheid waarmee resultaten geladen worden tijdens het plannen	2.47%	2	7.41%	6	9.88%	8	7.41%	6	16.05%	13	33.33%	27	23.46%	19
4	Snelheid waarmee resultaten geladen worden tijdens een reis	2.47%	2	16.05%	13	14.81%	12	13.58%	11	13.58%	11	24.69%	20	14.81%	12

Showing Rows: 1 - 4 Of 4

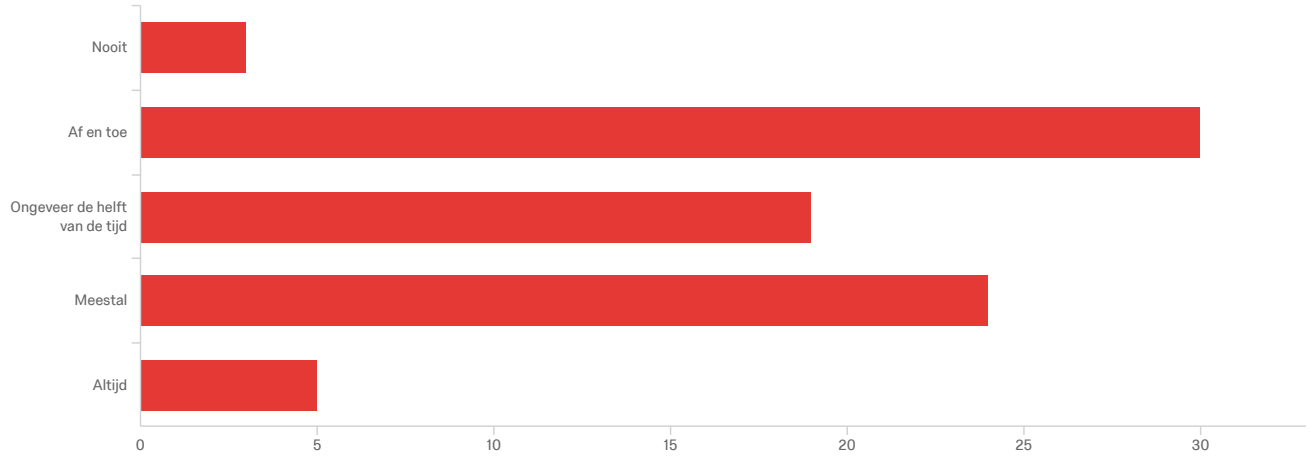
Q16 - Hoe tevreden ben je over het mobiele netwerk tijdens een treinreis?



#	Field		Choice Count
1	Zeer ontevreden	9.88%	8
2	Redelijk ontevreden	27.16%	22
3	Licht ontevreden	23.46%	19
4	Neutraal	11.11%	9
5	Licht tevreden	16.05%	13
6	Redelijk tevreden	11.11%	9
7	Zeer tevreden	1.23%	1
			81

Showing Rows: 1 - 8 Of 8

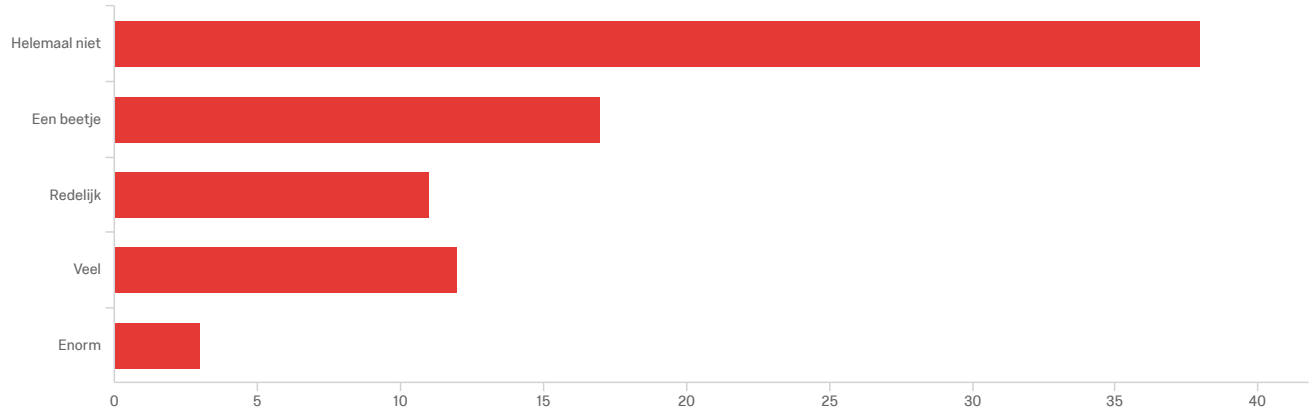
Q17 - Heb je soms last van een zeer trage of afwezige netwerkverbinding wanneer je op de trein zit, waardoor webpagina's enorm traag of zelfs niet laden?



#	Field		Choice Count
1	Nooit	3.70%	3
2	Af en toe	37.04%	30
3	Ongeveer de helft van de tijd	23.46%	19
4	Meestal	29.63%	24
5	Altijd	6.17%	5
			81

Showing Rows: 1 - 6 Of 6

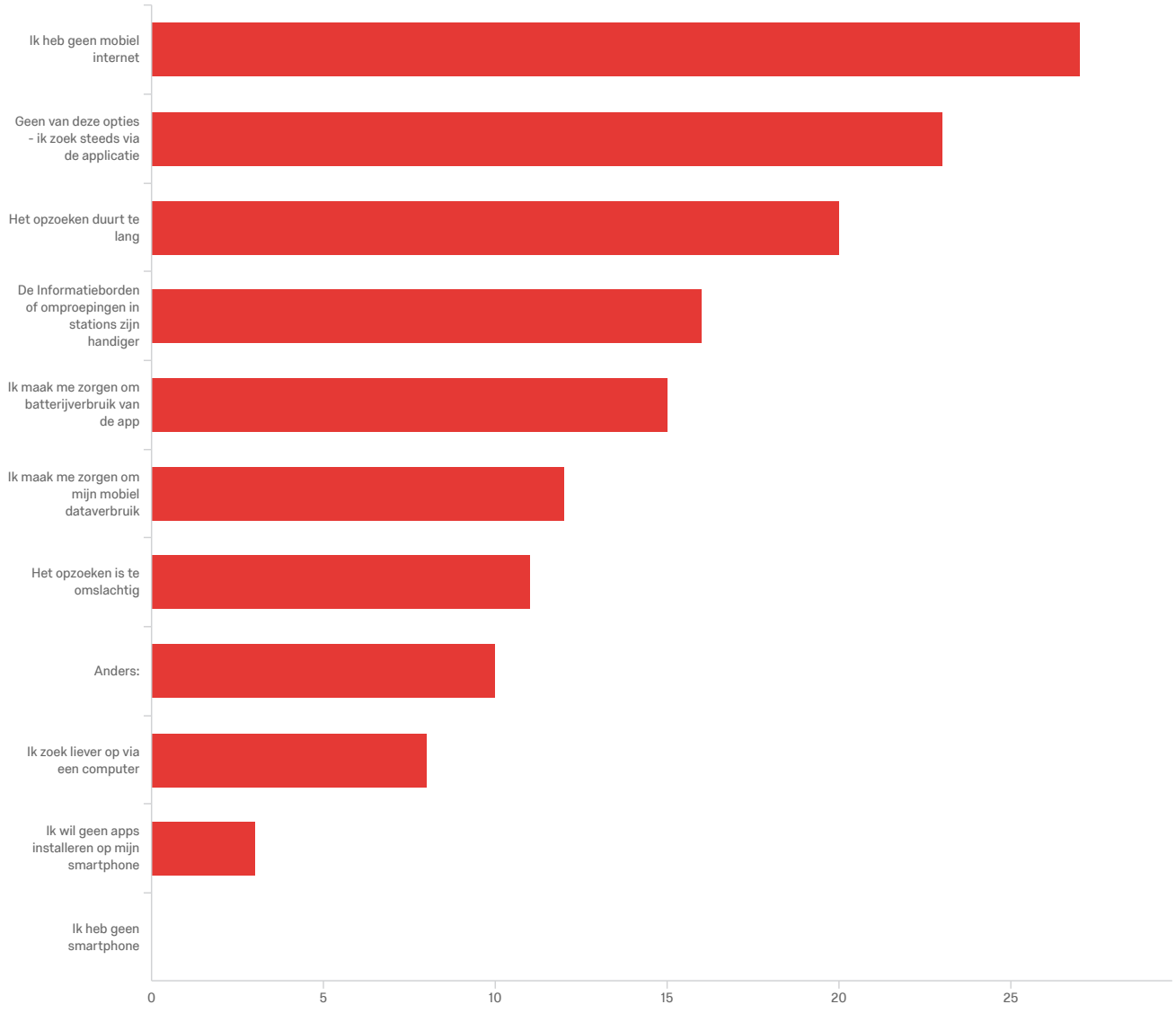
Q18 - Heb je schrik om meer mobiele data te verbruiken dan in je gsm abonnement of prepaid-bundel zit?



#	Field		Choice Count
1	Helemaal niet	46.91%	38
2	Een beetje	20.99%	17
3	Redelijk	13.58%	11
4	Veel	14.81%	12
5	Enorm	3.70%	3
			81

Showing Rows: 1 - 6 Of 6

Q19 - Als je informatie over treinen wenst en deze niet opzoekt via een applicatie, wat is hiervoor dan de reden? Markeer alle toepasselijke antwoorden.

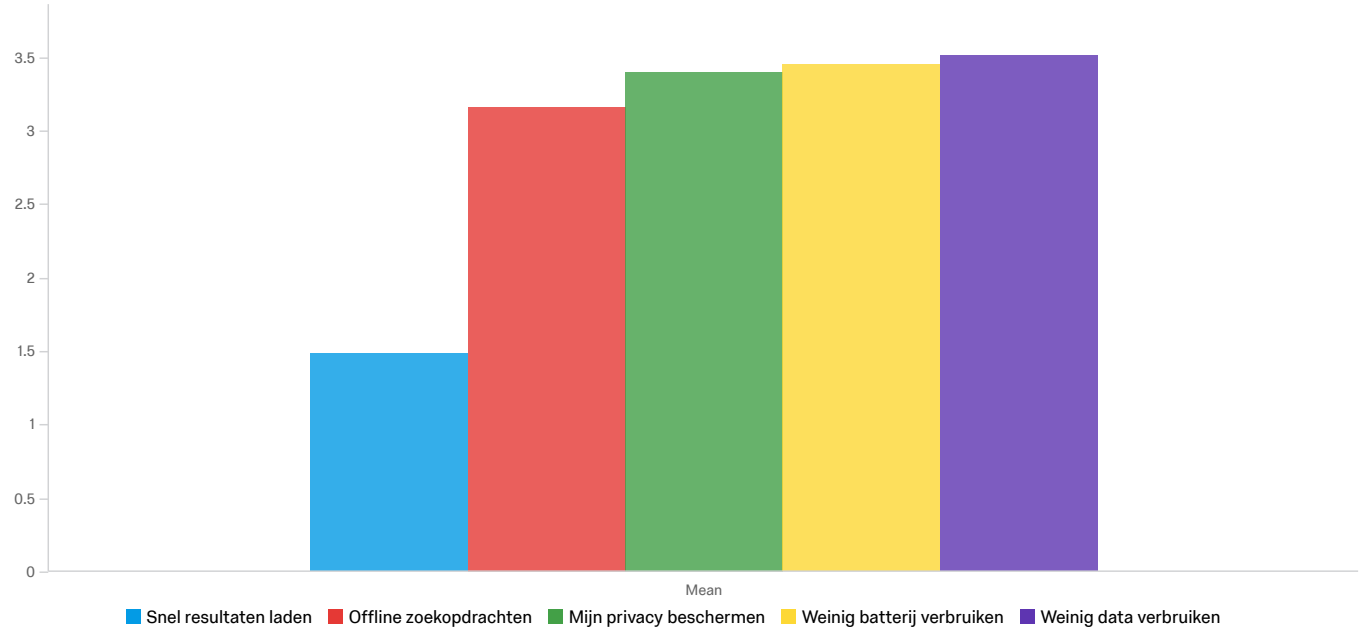


#	Field	Choice Count	
1	Het opzoeken duurt te lang	13.79%	20
2	Het opzoeken is te omslachtig	7.59%	11
3	Ik heb geen mobiel internet	18.62%	27
4	Ik maak me zorgen om mijn mobiel dataverbruik	8.28%	12
5	Ik maak me zorgen om batterijverbruik van de app	10.34%	15

6	Ik wil geen apps installeren op mijn smartphone	2.07%	3
7	Ik heb geen smartphone	0.00%	0
8	De Informatieborden of omproepingen in stations zijn handiger	11.03%	16
9	Ik zoek liever op via een computer	5.52%	8
10	Geen van deze opties - ik zoek steeds via de applicatie	15.86%	23
11	Anders:	6.90%	10
			145

Showing Rows: 1 - 12 Of 12

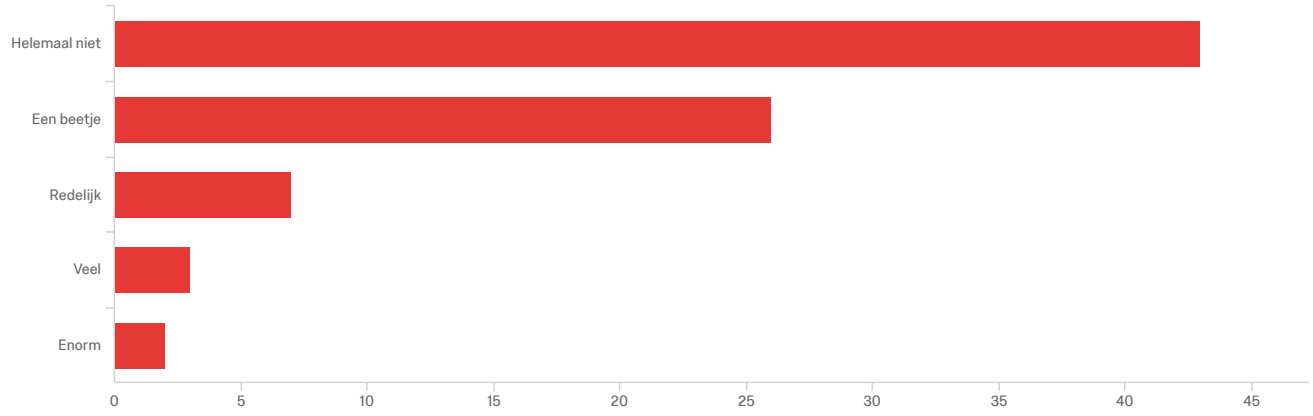
Q20 - Hoe belangrijk vind je onderstaande zaken in een app voor openbaar vervoer per trein? Rangschik van meest interessant naar minst interessant.



#	Field	1		2		3		4		5		Total
1	Offline zoekopdrachten	6.49%	5	29.87%	23	28.57%	22	11.69%	9	23.38%	18	77
2	Weinig data verbruiken	5.19%	4	18.18%	14	20.78%	16	32.47%	25	23.38%	18	77
3	Snel resultaten laden	67.53%	52	20.78%	16	7.79%	6	3.90%	3	0.00%	0	77
4	Mijn privacy beschermen	15.58%	12	10.39%	8	22.08%	17	22.08%	17	29.87%	23	77
5	Weinig batterij verbruiken	5.19%	4	20.78%	16	20.78%	16	29.87%	23	23.38%	18	77

Showing Rows: 1 - 5 Of 5

Q22 - Hoe bezorgd ben je om je privacy bij het gebruik van [QID3-ChoiceGroup-SelectedChoicesTextEntry]?

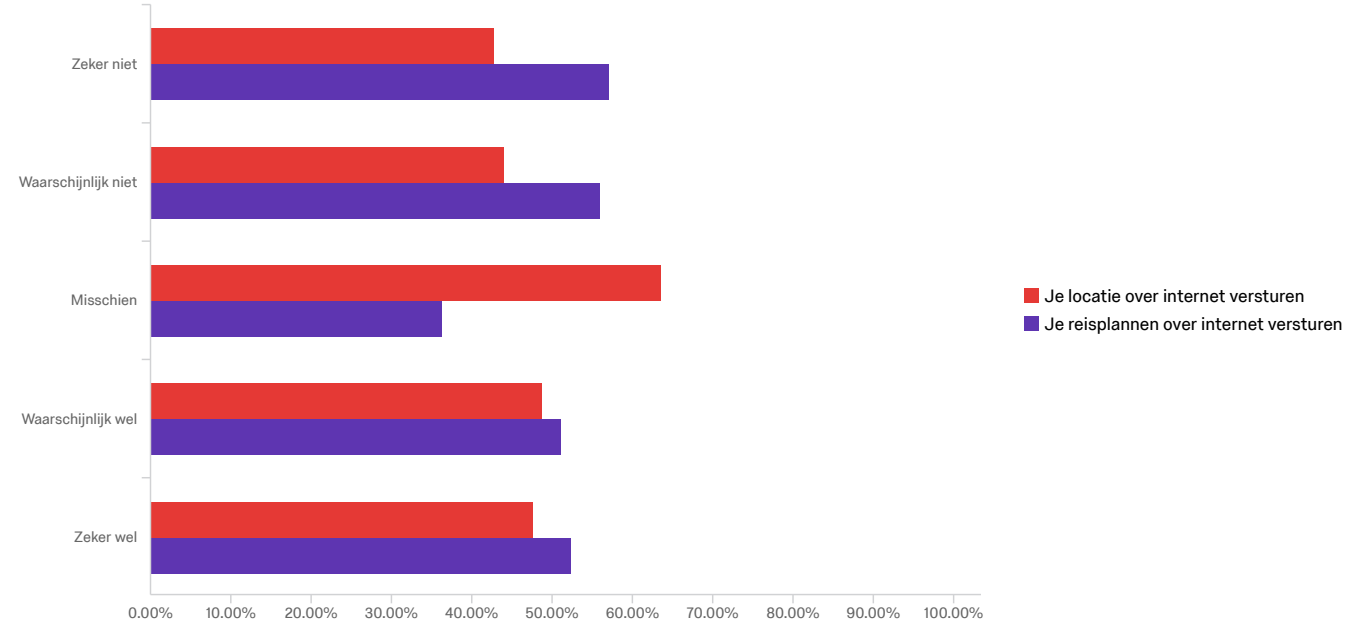


#	Field		Choice Count
1	Helemaal niet	53.09%	43
2	Een beetje	32.10%	26
3	Redelijk	8.64%	7
4	Veel	3.70%	3
5	Enorm	2.47%	2
			81

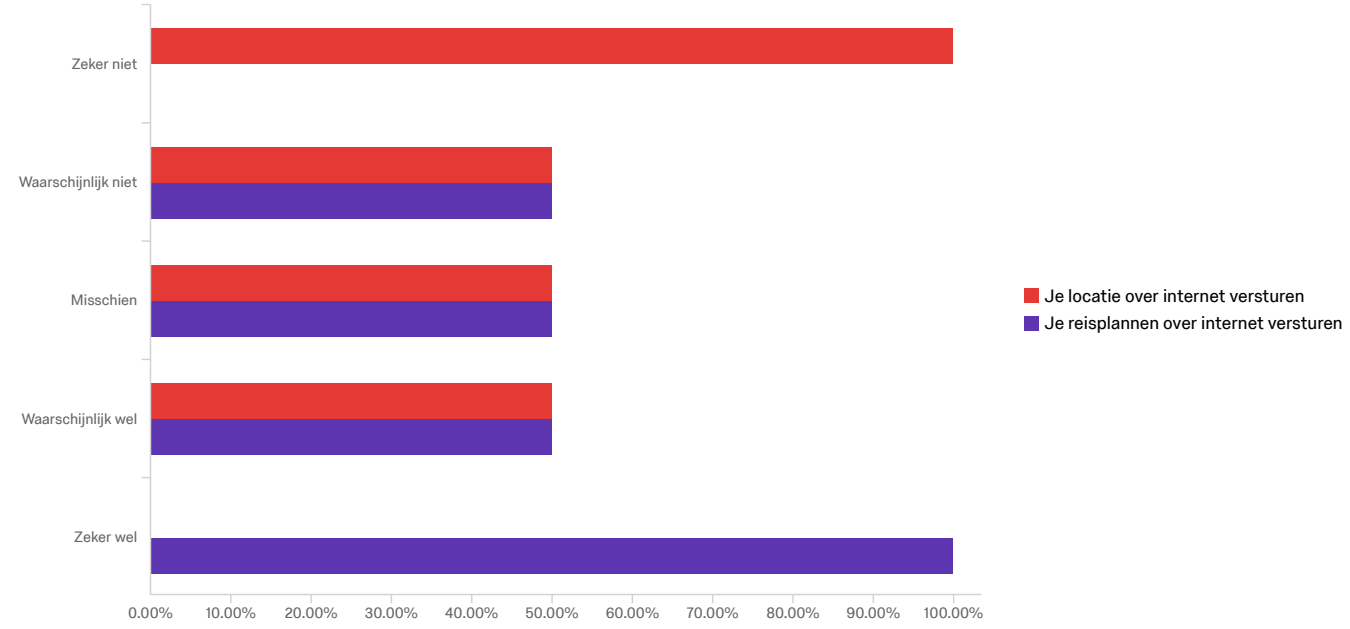
Showing Rows: 1 - 6 Of 6

Q23 - Denk je dat [QID3-ChoiceGroup-SelectedChoicesTextEntry] volgende zaken doet?

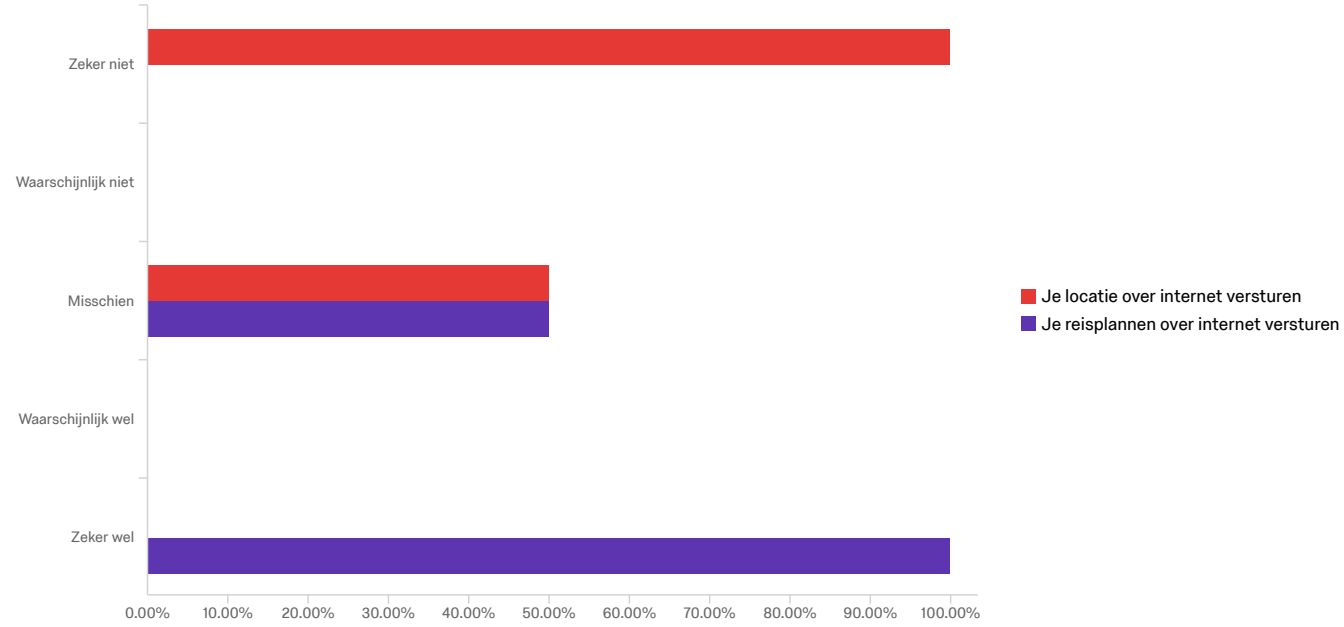
de officiële NMBS app



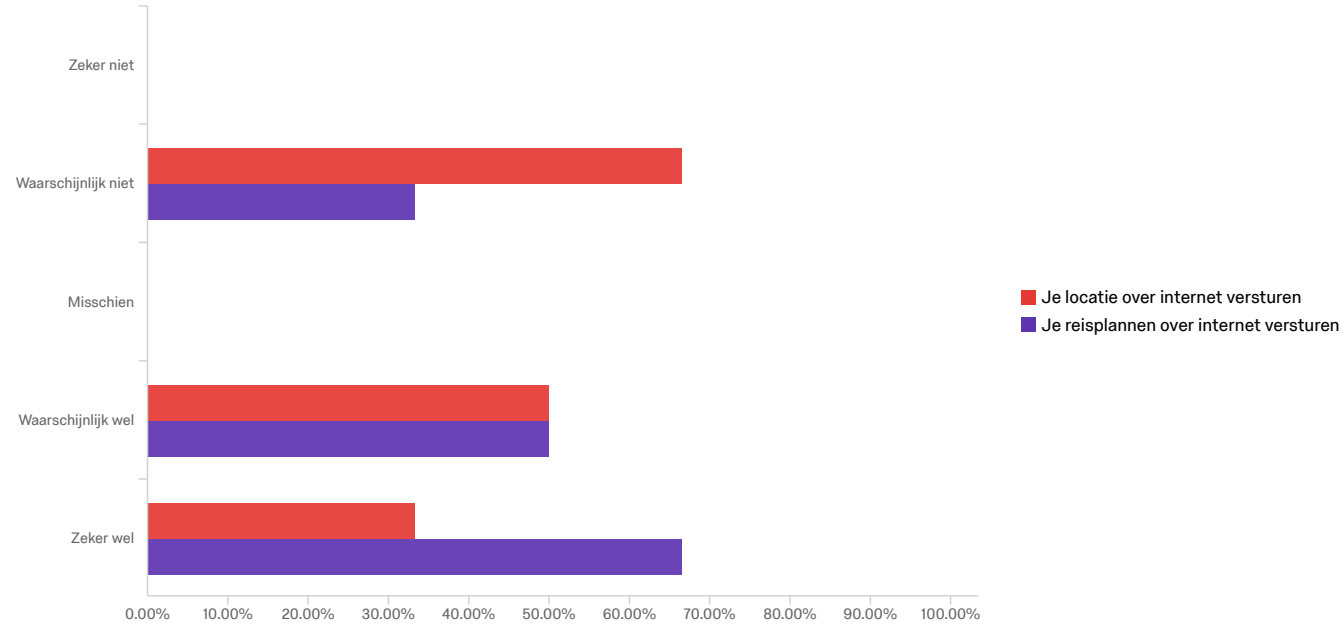
Hyperrail



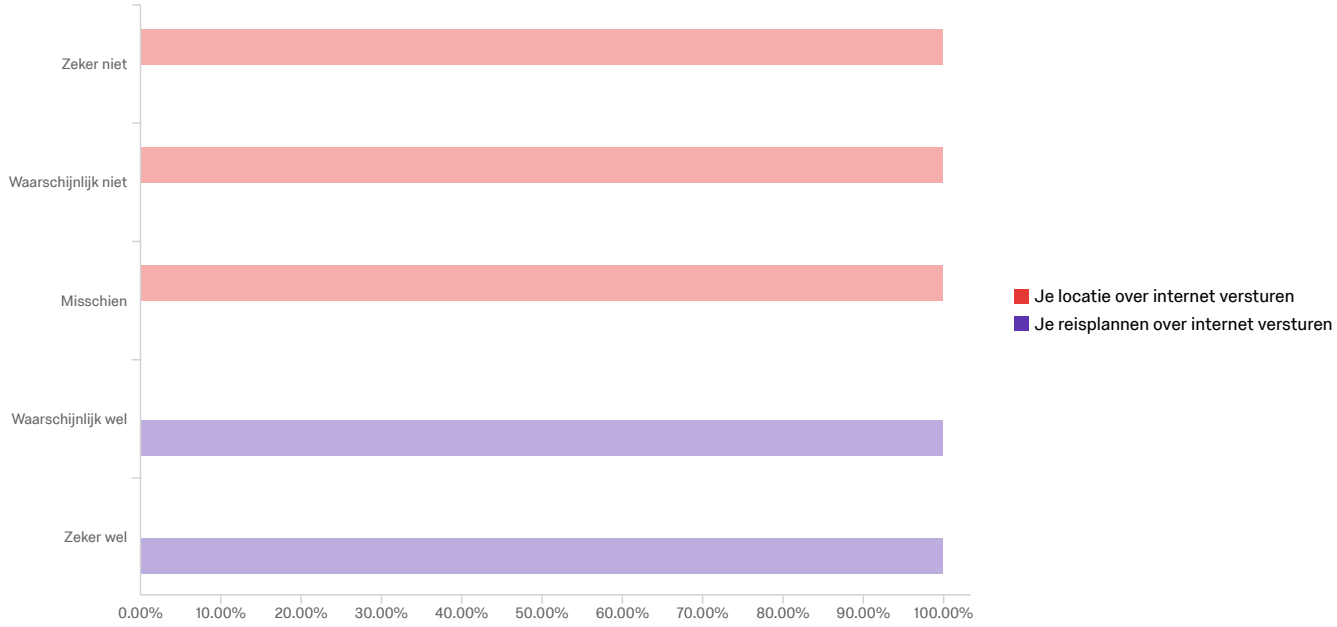
BeTrains



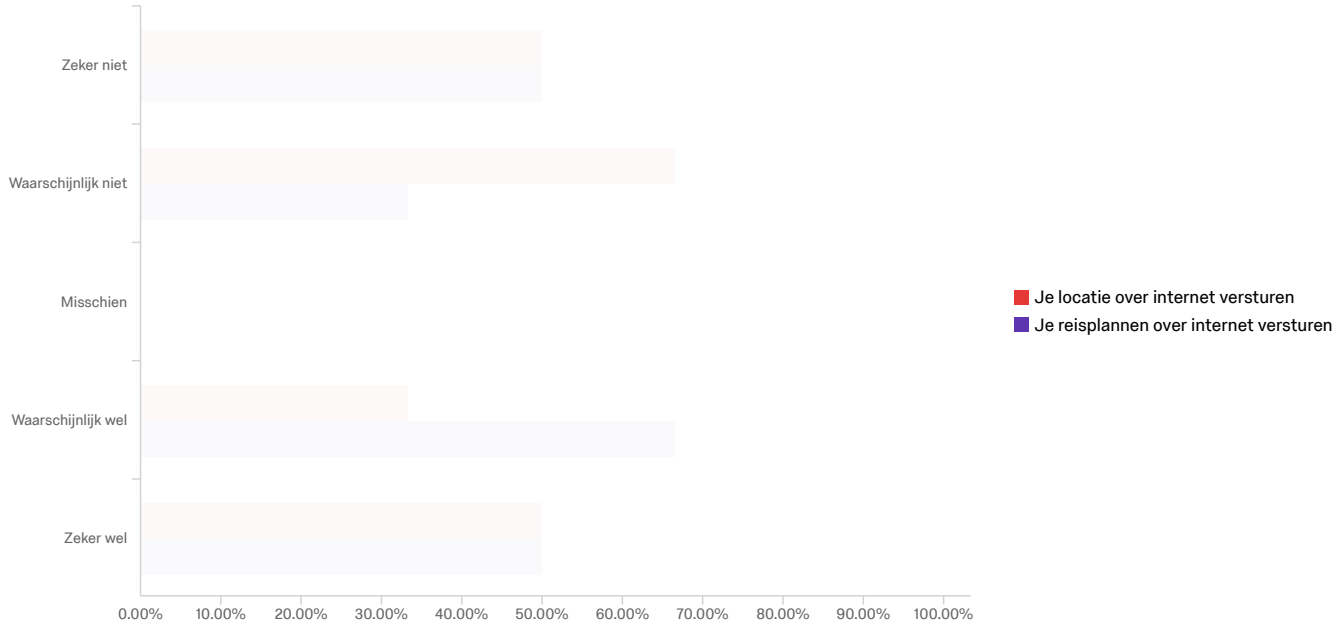
Railer



iRail webapplicatie



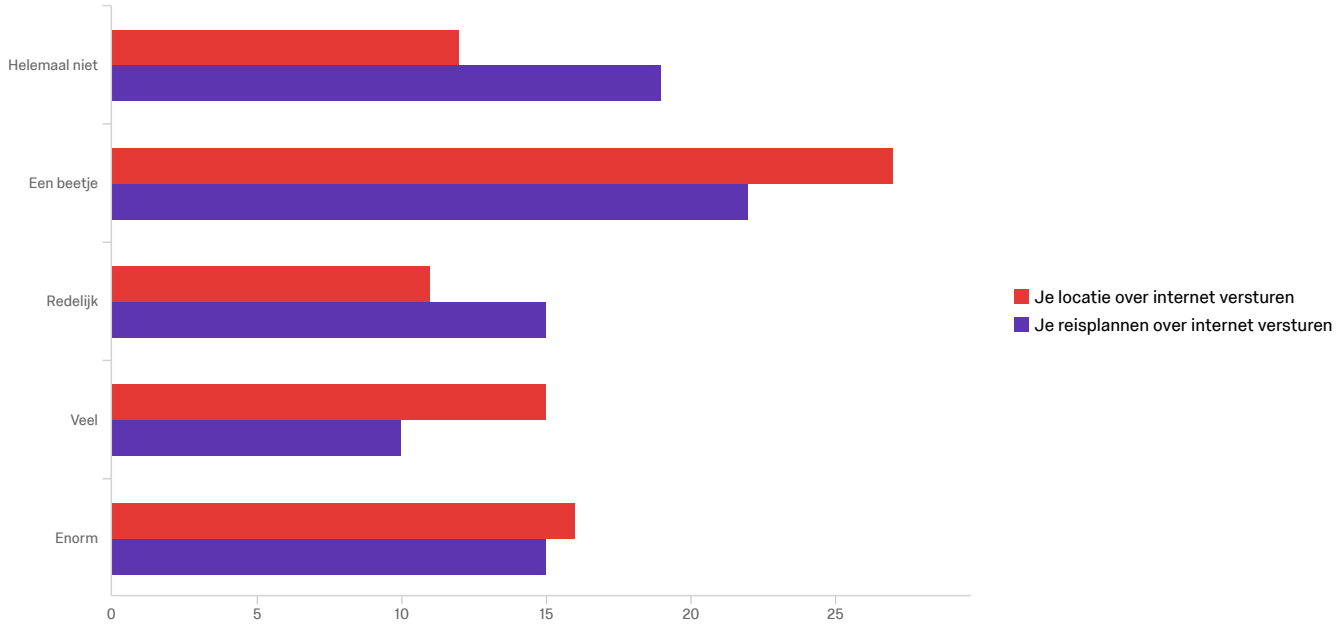
Anders:



#	Field	Zeker niet	Waarschijnlijk niet	Misschien	Waarschijnlijk wel	Zeker wel	Total
1	Je locatie over internet versturen	12.35% 10	20.99% 17	22.22% 18	29.63% 24	14.81% 12	81
2	Je reisplannen over internet versturen	6.17% 5	20.99% 17	13.58% 11	34.57% 28	24.69% 20	81

Showing Rows: 1 - 2 Of 2

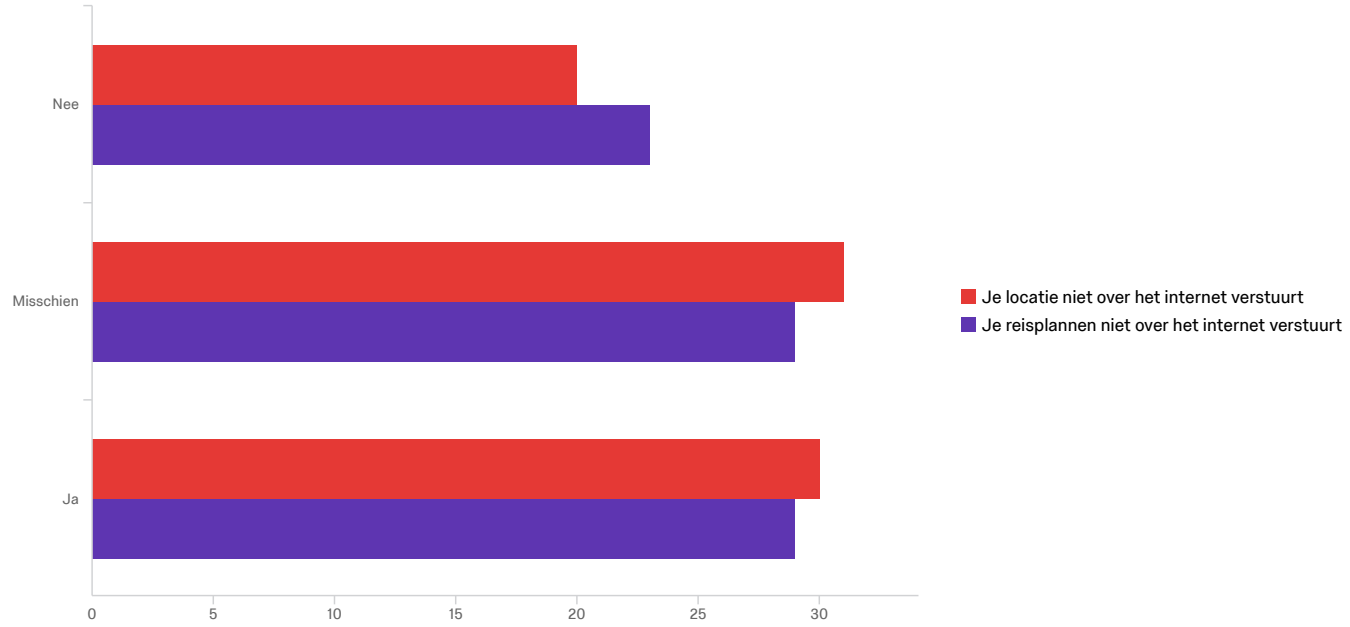
Q24 - Zou het je storen als [QID3-ChoiceGroup-SelectedChoicesTextEntry] volgende zaken doet?



#	Field	Helemaal niet	Een beetje	Redelijk	Veel	Enorm	Total
1	Je locatie over internet versturen	14.81% 12	33.33% 27	13.58% 11	18.52% 15	19.75% 16	81
2	Je reisplannen over internet versturen	23.46% 19	27.16% 22	18.52% 15	12.35% 10	18.52% 15	81

Showing Rows: 1 - 2 Of 2

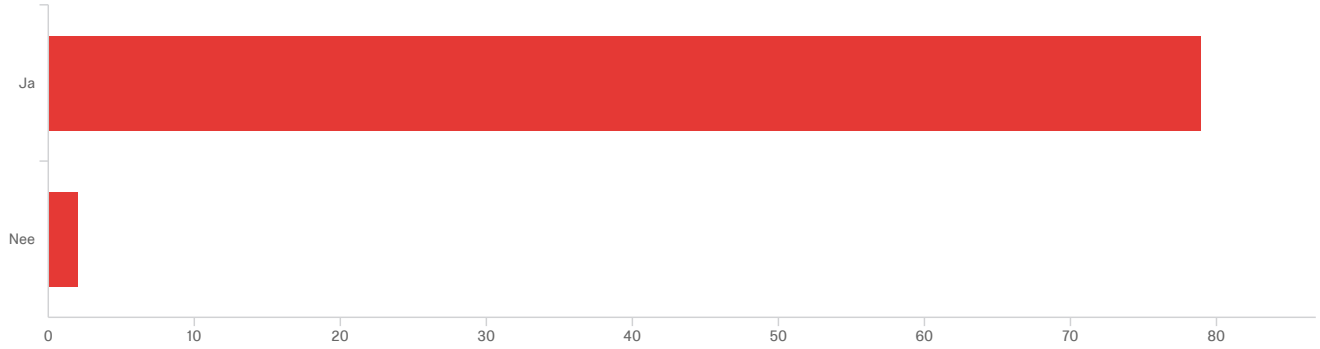
Q25 - Zou je overschakelen van [QID3-ChoiceGroup-SelectedChoicesTextEntry] naar een andere app, als deze andere app ...



#	Field	Nee		Misschien		Ja		Total
1	Je locatie niet over het internet verstuurt	24.69%	20	38.27%	31	37.04%	30	81
2	Je reisplannen niet over het internet verstuurt	28.40%	23	35.80%	29	35.80%	29	81

Showing Rows: 1 - 2 Of 2

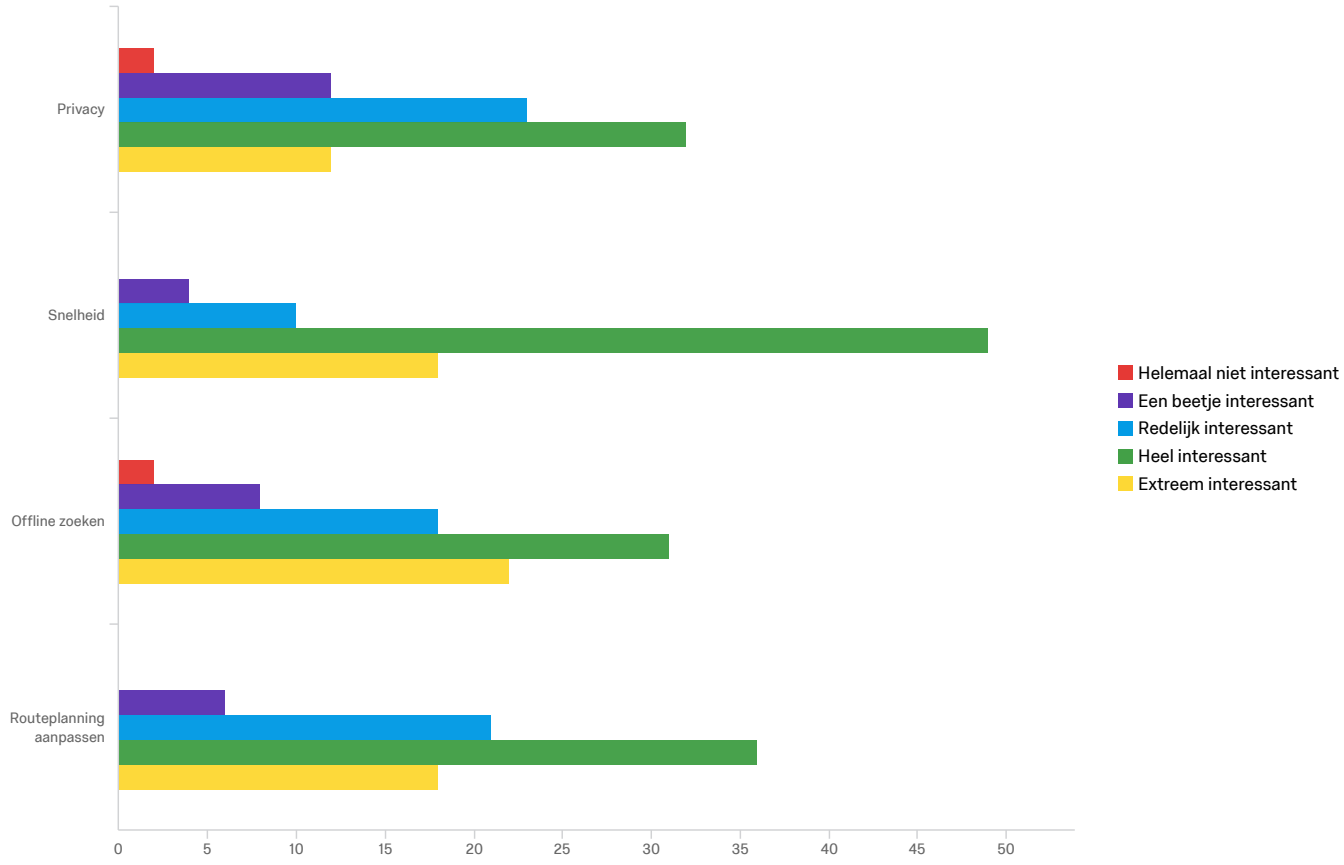
Q45 - Nu je meer weet over de verschillen ten opzichte van bestaande oplossingen, zijn we benieuwd in welke mate je geïnteresseerd bent in de nieuwe mogelijkheden van deze techniek. Heb je de uitleg op de vorige pagina volledig begrepen?



#	Field		Choice Count
1	Ja	97.53%	79
2	Nee	2.47%	2
			81

Showing Rows: 1 - 3 Of 3

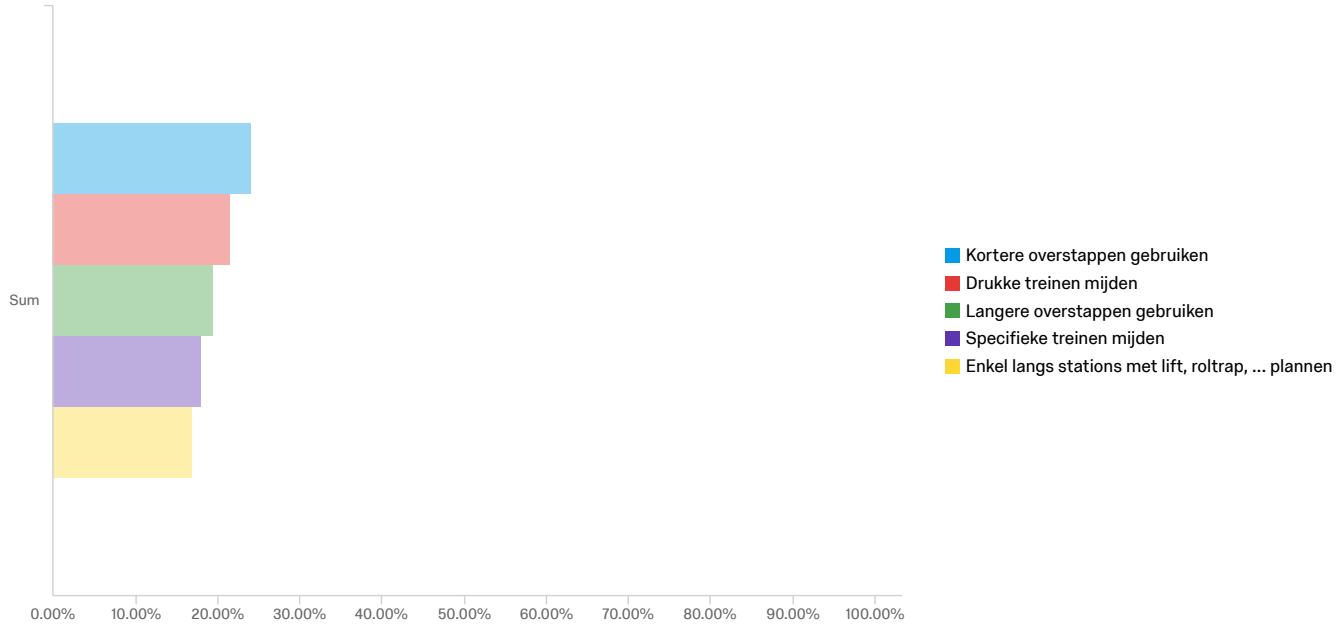
Q46 - Hoe interessant vind je deze aspecten?



#	Field	Helemaal niet interessant	Een beetje interessant	Redelijk interessant	Heel interessant	Extreem interessant	Total
1	Privacy	2.47% 2	14.81% 12	28.40% 23	39.51% 32	14.81% 12	81
2	Snelheid	0.00% 0	4.94% 4	12.35% 10	60.49% 49	22.22% 18	81
3	Offline zoeken	2.47% 2	9.88% 8	22.22% 18	38.27% 31	27.16% 22	81
4	Routeplanning aanpassen	0.00% 0	7.41% 6	25.93% 21	44.44% 36	22.22% 18	81

Showing Rows: 1 - 4 Of 4

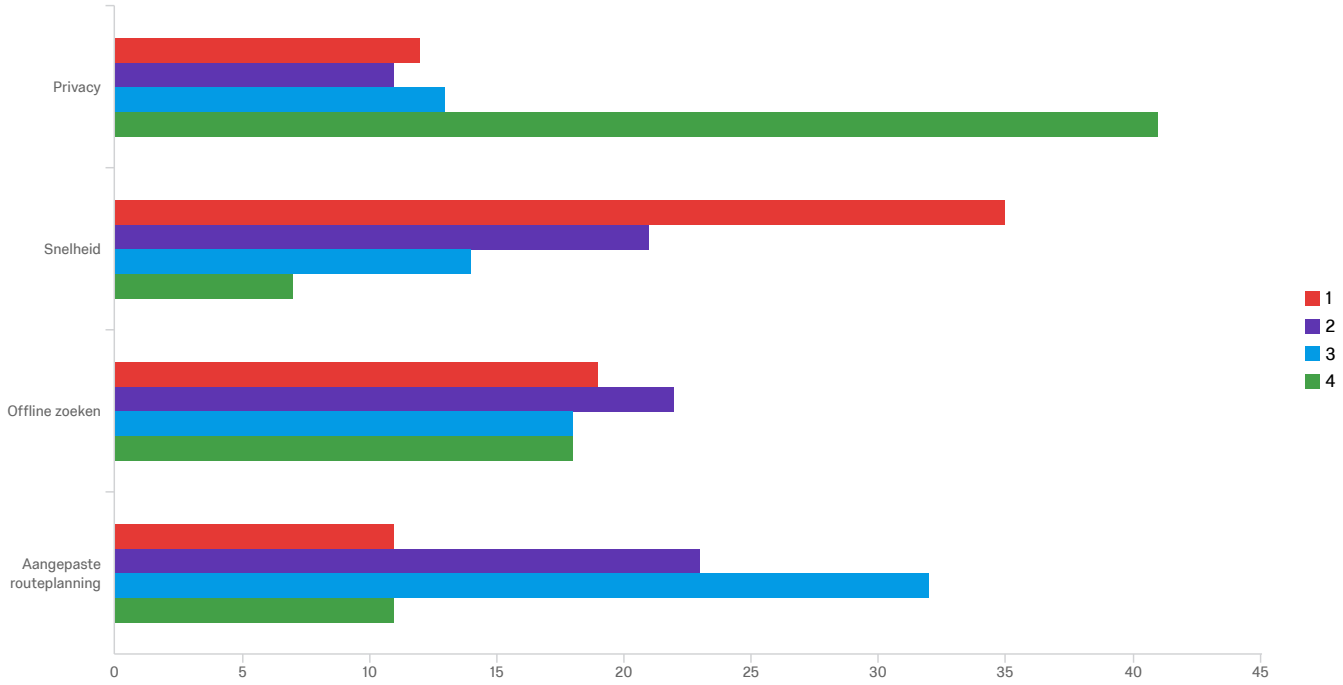
Q47 - Stel dat je in een app de routeplanning kon aanpassen. Hoe interessant zou je het vinden om ook deze parameters in te kunnen stellen?



#	Field	Helemaal niet interessant	Een beetje interessant	Redelijk interessant	Heel interessant	Extreem interessant	Total
1	Drukke treinen mijden	4.94% 4	29.63% 24	20.99% 17	33.33% 27	11.11% 9	81
2	Specifieke treinen mijden	17.28% 14	33.33% 27	22.22% 18	22.22% 18	4.94% 4	81
3	Kortere overstappen gebruiken	3.70% 3	16.05% 13	20.99% 17	41.98% 34	17.28% 14	81
4	Langere overstappen gebruiken	16.05% 13	22.22% 18	27.16% 22	29.63% 24	4.94% 4	81
5	Enkel langs stations met lift, roltrap, ... plannen	34.57% 28	16.05% 13	22.22% 18	22.22% 18	4.94% 4	81

Showing Rows: 1 - 5 Of 5

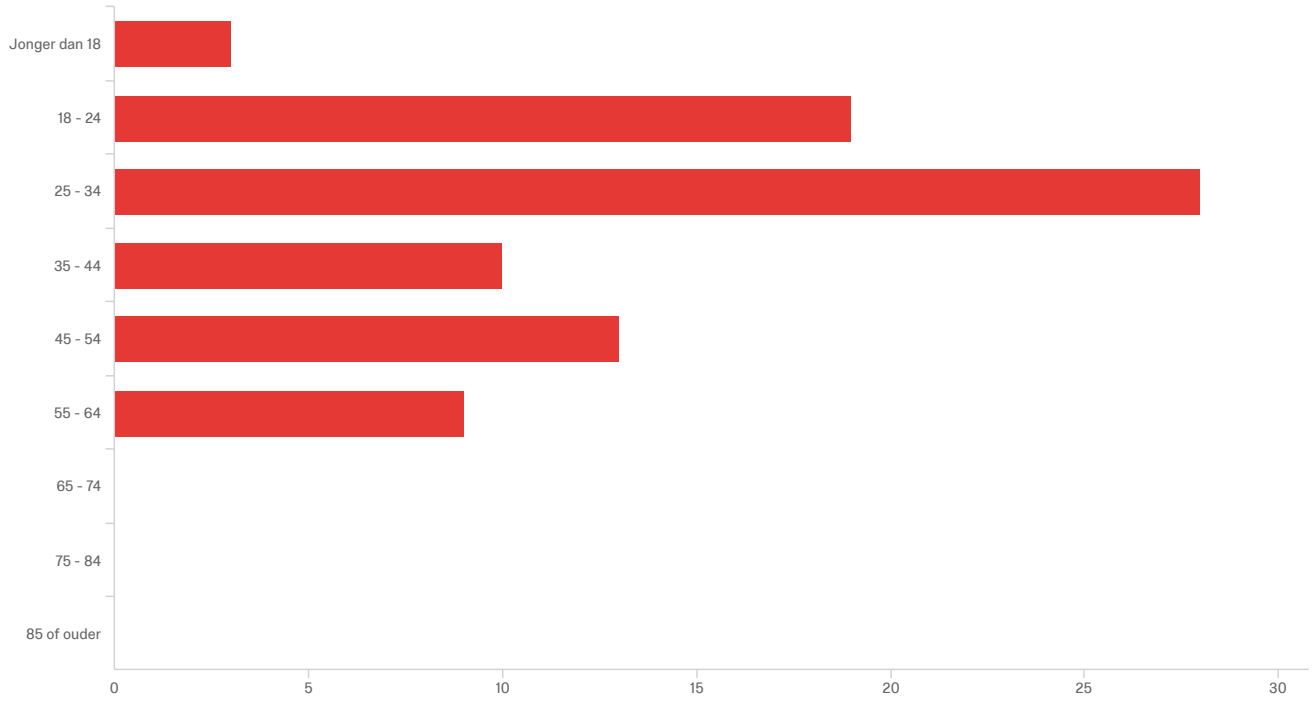
Q48 - Rangschik de volgende aspecten van Linked Connections van meest interessant naar minst interessant.



#	Field	1		2		3		4		Total
1	Privacy	15.58%	12	14.29%	11	16.88%	13	53.25%	41	77
2	Snelheid	45.45%	35	27.27%	21	18.18%	14	9.09%	7	77
3	Offline zoeken	24.68%	19	28.57%	22	23.38%	18	23.38%	18	77
4	Aangepaste routeplanning	14.29%	11	29.87%	23	41.56%	32	14.29%	11	77

Showing Rows: 1 - 4 Of 4

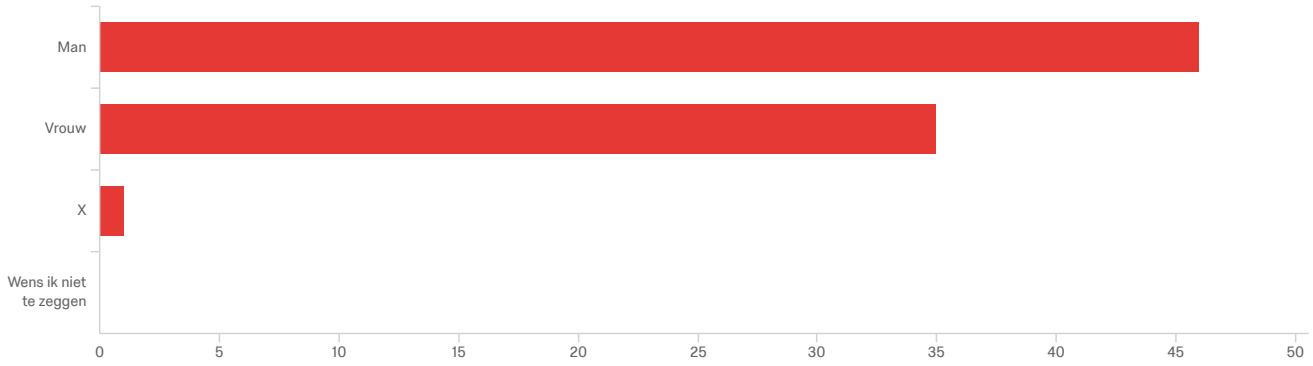
Q49 - Hoe oud ben je?



#	Field		Choice Count
1	Jonger dan 18	3.66%	3
2	18 - 24	23.17%	19
3	25 - 34	34.15%	28
4	35 - 44	12.20%	10
5	45 - 54	15.85%	13
6	55 - 64	10.98%	9
7	65 - 74	0.00%	0
8	75 - 84	0.00%	0
9	85 of ouder	0.00%	0
			82

Showing Rows: 1 - 10 Of 10

Q50 - Wat is je geslacht?



#	Field		Choice Count
1	Man	56.10%	46
2	Vrouw	42.68%	35
3	X	1.22%	1
4	Wens ik niet te zeggen	0.00%	0
			82

Showing Rows: 1 - 5 Of 5

End of Report