The user perceived performance of route planning APIs

Bert Marcelis

Supervisor: Prof. dr. ir. Ruben Verborgh
Counsellors: Pieter Colpaert, Julian Andres Rojas Melendez

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Information Engineering Technology

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Koen De Bosschere
Faculty of Engineering and Architecture
Academic year 2017-2018

Dankwoord

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van listings

"Inspirational quote"

~Source

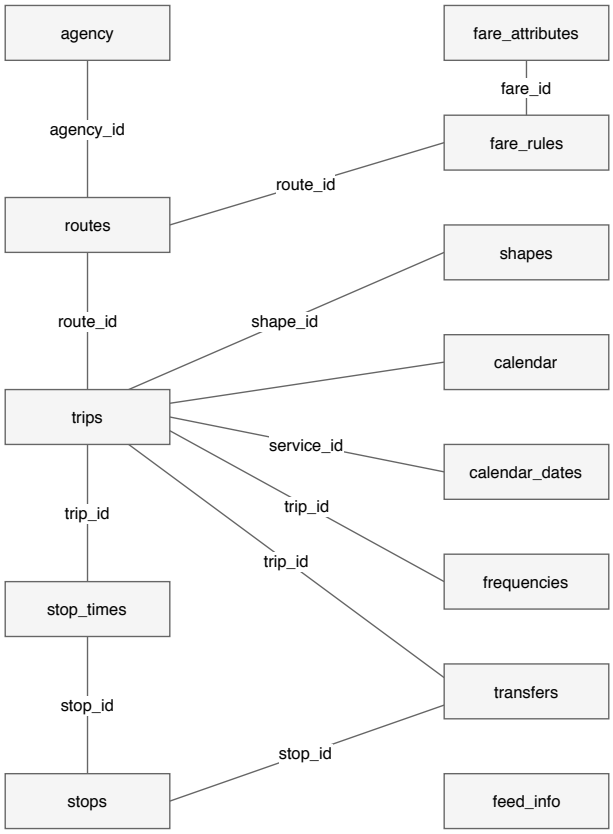
1 Inleiding

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) die gebruikers informatie verstrekken over vertrekken, aankomsten, ritten, routes en vertragingen. Voorbeelden hiervan in België zijn iRail.be, HyperRail en Railer, 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 APIs 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 dag. Om al deze data compact op te slaan en te versturen, worden deze opgeslagen in de vorm van regels. Deze regels omschrijven wanneer welk voertuig welke rit maakt. Om op basis van deze regels vragen te beantwoorden, dient deze set abstracte regels omgevormd te worden naar een gepast model waarin ritten en stopplaatsen opgevraagd kunnen worden, en routes berekend kunnen worden. Hiervoor zijn, afhankelijk van welke informatie gewenst is, zware berekeningen vereist, die afhankelijk van de grootte van het GTFS bestand vijf à tien

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

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

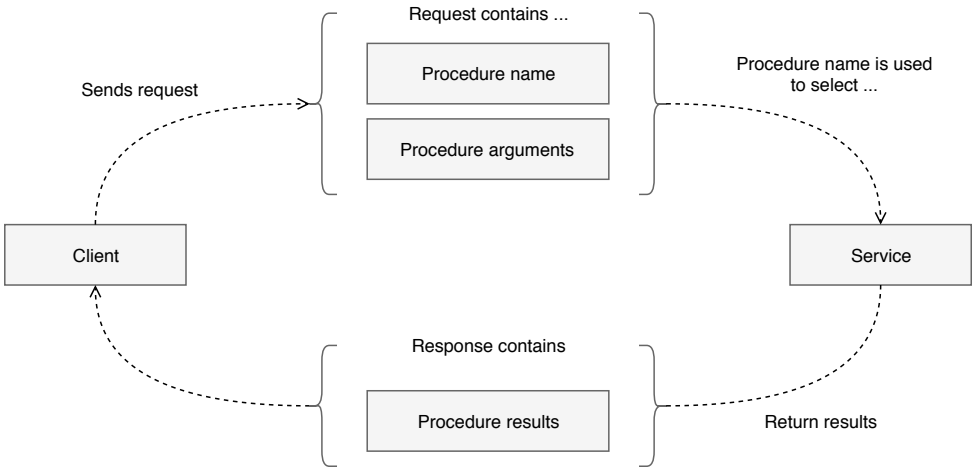


Figuur 1.1: De bestandsstructuur van GTFS data.

minuten kunnen duren op een moderne computer. Gebruikers kunnen geen 10 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 vervoersdata: 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 vervoersdata te delen met grote bedrijven, en in mindere mate voor individuele ontwikkelaars die vervoers data eenvoudig willen visualiseren (digital signage, routeplanner applicaties, websites, ...).

Anderzijds zijn er traditionele Remote Procedure Call (RPC) zoals iRail[3], die beschikken over verschillende endpoints die specifieke vragen kunnen beantwoorden. Achterliggend kunnen zware berekeningen uitvoeren of grote databases raadplegen zonder dat de gebruiker hier nadeel van ondervindt. Deze antwoorden zijn rechtstreeks bruikbaar voor de client toepassing, maar bieden enkel een antwoord op één specifieke vraag. Een andere vraag, al dan niet door dezelfde client, vereist een nieuwe request naar de server, en zal een ander antwoord tot gevolg hebben. Elk verzoek naar de server vraagt relatief veel processortijd langs de serverkant. Een continue internetverbinding is dus vereist, en server-side is een potentieel grote en dure infrastructuur

[3]https://irail.be

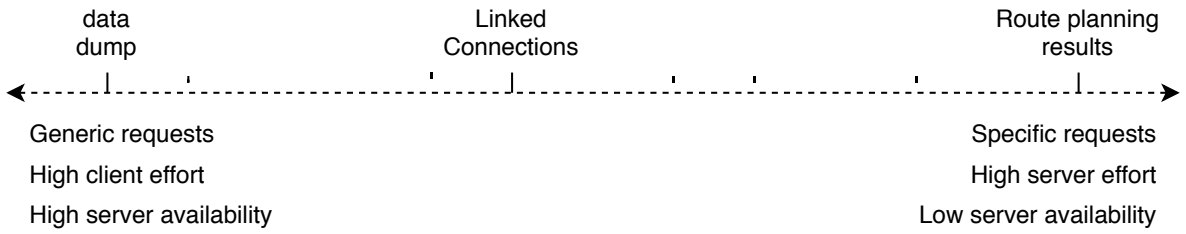


Figuur 1.2: De werkwijze van een RPC API.

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 iemand een API aanbiedt die achterliggend door meerdere datasets zoekt. Simpelweg twee API's combineren is niet 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. 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 [3]. In deze figuur is de as toegepast op HTTP interfaces voor routeplanning [4].

1.1 Wat is user-perceived performance?

Elke interface voor het ophalen van data heeft specifieke eigenschappen zoals latency, performance, cache hergebruik, ... [5]. Wanneer we verschillende technieken vergelijken door dezelfde user-agent, 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 gedefinieerd in 2000 door Roy T. Fielding gedefinieerd als de tijd tussen het selecteren van een link en het renderen van een bruikbaar resultaat [6]. Latency treedt op 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 te antwoorden
5. de tijd die nodig is om het antwoord te verzenden
6. de tijd voor 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 [6].

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 [7]. Een ander gebruik van de term focust op de niet-instrumentele norden en ervaringen

in een complexere zin [7]. In beide gevallen is UX een centraal belang in Human-Computer Interfaces (HCI) [8]. 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 [9]. 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 in de user experience [10]. Een applicatie presteert beter 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 [11].

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 queries te beantwoorden door middel van een lineair groeiende lijst van connecties [4]. Bovendien gebruiken alle user-agents dezelfde lijst, waardoor deze zeer cachebaar is. Bij stijgende belasting daalt de tijd die nodig is per verzoek [12].

Terwijl de cost-efficiency van Linked Connections reeds is aangetoond, waarbij Linked Connections hetzelfde aantal verzoeken kan beantwoorden met slechts 25% van de rekencapacteit[12, 13], is er nog geen onderzoek gebeurt 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 [12] en de originele Linked Connections webserver. Als user-agent zullen we een fork van de Android HyperRail[6] applicatie gebruiken, gemodificeerd om de genoemde API's te gebruiken. Door deze testopstelling zijn de oorspronkelijke data, de server hardware, de user-agent en de client hard-

[6]https://hyperrail.be

ware gelijk 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) [14, 15, 16]. Dit algoritme vereist een op vertrektijd gesorteerde lijst van vertrekken. Dit is de exacte definitie van de LinkedConnections 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-experienced performance onderzocht worden. Hiertoe worden begeleide user tests gehouden, waarbij een aantal testgebruikers 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 er technische data verzameld worden, 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 vervoer maatschappijen die gebruik maken van tijdsschema's, ongeacht het soort voertuig dat gebruikt wordt.

1.3 Onderzoeksvraag

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?

Tijdens treinreizen valt bij veel mensen de mobiele netwerkverbinding regelmatig weg. Dit wordt

ook gestaafd door de resultaten van de enquete die in het kader van dit onderzoek werd uitgevoerd, terug te vinden in hoofdstuk 4. 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 De gebruiker ervaart de mogelijkheid voor offline zoekopdrachten als een meerwaarde.

Uit gesprekken met meerdere treinreizigers blijkt dat een aantal reizeigers 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 kunnen terug te vinden. We vermoeden ook dat minder mobiele gebruikers 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 [11]. 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 API als de server-side API 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

In plaats van een dump van planningsdata of een volledige routeplanner te publiceren, publiceert Linked Connections zogenoemde connecties. Een connectie is het kleinste ondeelbaar stuk van een treinrit, en beschrijft het vertrek in een station en de aankomst in het volgence station op de route. Deze connecties worden 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 [17].

[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. Codefragmenten 1 en 2 tonen een ingekort resultaat voor de vertrekken bij de NMBS op 20 maart 2018, 12:30. De volledige specificatie kan teruggevonden worden op de LC website[4].

Verzoek: *https://graph.irail.be/sncb/connections?departureTime=2018-03-20T12:30:00.000Z*

Een voorbeeld van een LC pagina zien we in fragmenten 1 en 2, met volgende data:

1. *@context*: Deze lijst, zichtbaar in fragment 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 fragment2, 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 aakomst.
   (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

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

```
"@context": {
 "xsd": "http://www.w3.org/2001/XMLSchema#",
"lc": "http://semweb.mmlab.be/ns/linkedconnections#",
"hydra": "http://www.w3.org/ns/hydra/core#",
"gtfs": "http://vocab.gtfs.org/terms#",
"Connection": "lc:Connection",
"...": "..."
},
"@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 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: Voorbeeld Linked Connections Formaat: graph

2.2 Algoritmes

2.2.1 Connection Scan Algoritme

Routeplanning wordt vaak opgelost met behulp van (een variant op) het algoritme van Dijkstra [14, 18, 19]. Toepassingen die gebruik maken van Dijkstra vereisen echter een graaf en 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 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.

Gezien het Belgische spoortnetwerk relatief klein is in vergelijking met andere landen, is CSA ruim snel genoeg. In de praktijk blijkt de standaard implementatie van CSA snel genoeg voor realtime opzoekingen, ook voor grote spoornetwerken zoals het Duitse net[15]. 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[15, 16]. 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 [18].

Het Connection Scan Algoritme (CSA) werd voor het eerst beschreven door Ben Strasser in 2013 [14]. Dit algoritme vereist van een lijst met vertrekken gesorteerd op vertrektijd. Hiermee worden alle routes in een tijdsinterval efficiënt berekend [15, 16]. In tegenstelling tot Dijkstra's algoritme is er geen graaf of *priority queue* benodigd. 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 [15]. In de praktijk is het vooral belangrijk om snel rekening te kunnen houden met vertragingen bij vertrek of aankomst [15, 16]. CSA berekent oorspronkelijk 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 [15].

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 [19, 16]. Door het aantal overstappen te optimaliseren voorkomen we dat van eenzelfde trein wordt afgestapt om later terug op te stappen [15]. Naast de tijd van aankomst, zijn er nog een aantal andere criteria die vaak geoptimaliseerd worden. Het populairste tweede criterium is het aantal overstappen, gevolgd door de prijs [16]. Optimalisatie van de prijs is echter zeer complex vanwege de complexe tariefplannen bij openbaar vervoer[20]. Dit valt buiten de context van deze masterproef.

De werking en implementatie van CSA worden uitvoerig behandeld in [16]. Dit algoritme kan zonder veel wijzigingen geïmplementeerd worden in zowel Java[5] als PHP[6].

Wanneer dit algoritme geïmplementeerd wordt merken we echter duidelijke verschillen met de voorgestelde routes door de NMBS. Deze verschillen manifesteren zich vooral in de keuze van het station 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 betere manier aan *journey-extraction* te doen.

De implementatie en evolutie van het CSA algoritme worden uitgelegd aan de hand van code fragmenten in Java. Er wordt verondersteld dat sectie 4.2 van [16] 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.

Allereerst dienen we twee wijzigingen door te voeren aan de gegevensstructuren. De arrays S en T, waarin respectievelijk zogenoemde 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 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 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

[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`

```
class TrainTriple {
  /**
  * 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 3: In tegenstelling tot [16] wordt niet enkel de aankomsttijd, maar ook de afstaphalte en het aantal overstappen bijgehouden per trip.

die ofwel te vroeg, ofwel te laat vallen. Zoals te zien in fragment 5 stellen we een *flag* in wanneer we voorbij de vroegste vertrekdatum zijn. In dit geval zullen we na het overlopen van deze lijst geen nieuwe lijsten meer ophalen. Door deze methode toe te passen kunnen we nieuwe pagina's inladen wanneer deze nodig zijn, zonder te veel op voorhand in te moeten laden.

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 [16]. 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 [16, 19], 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 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 latere 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 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 genoeg tijd is om over te stappen, maar waarbij het aantal overstappen het maximum aantal niet overschrijdt. De intra-footpaths [16, 19], nodig voor het bepalen van de tijd die reiziger nodig heeft om over te stappen, worden ingesteld op een standaardwaarde die eventueel door de gebruiker gewijzigd kan worden. Wanneer we een overstap vinden die aan deze voorwaarden

```
class StationQuintuple {
/**
* 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 4: In tegenstelling tot [16] wordt niet enkel de vertrek- en aankomsttijd, maar ook het aantal overstappen en de afstaphalte van de volgende trein bijgehouden.

voldoet, verhogen we het aantal overstappen ook met één. Deze aanpak is eenvoudiger dan de array-gebaseerde aanpak omschreven in [16]. 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 [16] 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 door gebruikers ingesteld kan worden om de routes te personaliseren.

Bij het bepalen van de vroegste aankomsttijd (fragment 7), wordt nu ook het aantal overstappen dat bij deze aankomsttijd hoort bepaald, en de connectie waar van de trein afgestapt wordt. We geven bij gelijke aankomsttijden de voorkeur aan overstappen: aangezien de vertrekkende voertuigen volgens dalende vertrektijd overlopen worden, geven we dus de voorkeur aan zo vroeg

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

mogelijk overstappen. Dit geeft extra marge binnen de trip. 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 afstap halte 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 afstap halte (de connectie die op dit moment onderzocht wordt) meer tijd voor een overstap geeft. Indien dit het geval is, werken we de afstap halte bij. Het bijwerken van een bestaande trip is zichtbaar in fragment 9.

Het bijwerken van de stopprofielen, zichtbaar in fragment 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 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 is, en een beter alternatief voor het overlopen van de volledige lijst.

```
 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 6: Het aantal overstappen wordt bepaald bij het bepalen van minimale aankomsttijden

De lijst met volledige routes reconstrueren (fragment 11) is relatief eenvoudig. Voor elk profiel horend bij de stoplocatie 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 12). Wanneer deze gelijk zijn, hebben we het volgende deel van de reis gevonden.

```
// Determine T3, the time of arrival when taking the best possible transfer in
↪  this station
if (S.containsKey(connection.arrivalStationUri)) {
  int position = S.get(connection.arrivalStationUri).size() - 1;
  StationQuintuple quintuple = S.get(connection.arrivalStationUri).get(position);

  while (
    (quintuple.departureTime.minusSeconds(transferSeconds).getMillis() <=
    ↪  connection.arrivalTime.getMillis() ||
    quintuple.transfers >= maxTransfers) &&
    position > 0
  ) {
    position--;
    quintuple = S.get(connection.arrivalStationUri).get(position);
  }
  if (quintuple.departureTime.minusSeconds(transferSeconds)
      .isAfter(connection.arrivalTime) &&
      quintuple.transfers <= maxTransfers) {
    T3_transferArrivalTime =
    ↪  quintuple.arrivalTime.plusSeconds(extraTimeInsteadOfTransfer);
    // Using this transfer will increase the number of transfers with 1
    T3_transfers = quintuple.transfers + 1;
  } else {
    // When there isn't a reachable connection, transferring isn't an option
    T3_transferArrivalTime = infinite;
    T3_transfers = 999;
  }
} else {
  // When there isn't a reachable connection, transferring isn't an option
  T3_transferArrivalTime = infinite;
  T3_transfers = 999;
}
```

Code 7: Bij een eventuele overstap worden ook extra factoren in rekeningen gebracht.

```
DateTime Tmin;
LinkedConnection exitTrainConnection;
int numberOfTransfers;

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 8: Bepalen van de vroegste aankomsttijd

```
if (Tmin.isEqual(T.get(connection.trip).arrivalTime) &&
  T3_transferArrivalTime.isEqual(T2_stayOnTripArrivalTime) &&
  S.containsKey(T.get(connection.trip).arrivalConnection.arrivalStationUri) &&
  S.containsKey(connection.arrivalStationUri)
) {
  LinkedConnection currentTrainExit = T.get(connection.trip).arrivalConnection;

  StationQuintuple quint = new StationQuintuple();
  quint.departureTime = connection.departureTime;
  quint.departureConnection = connection;
  quint.arrivalTime = Tmin;

  // Current situation
  quint.arrivalConnection = currentTrainExit;
  Duration currentTransfer = new Duration(currentTrainExit.arrivalTime,
  ↪  getFirstReachableConnection(quint).departureTime);
  // New situation
  quint.arrivalConnection = exitTrainConnection;
  Duration newTransfer = new Duration(exitTrainConnection.arrivalTime,
  ↪  getFirstReachableConnection(quint).departureTime);

  if (newTransfer.isLongerThan(currentTransfer)) {
    TrainTriple triple = new TrainTriple();
    triple.arrivalTime = Tmin;
    triple.arrivalConnection = exitTrainConnection;
    triple.transfers = numberOfTransfers;
    T.put(connection.trip, triple);
  }
}

if (Tmin.isBefore(T.get(connection.trip).arrivalTime)) {
  TrainTriple triple = new TrainTriple();
  triple.arrivalTime = Tmin;
  triple.arrivalConnection = exitTrainConnection;
  triple.transfers = numberOfTransfers;
  T.put(connection.trip, triple);
}
```

Code 9: Bijwerken van de trips gegevensstructuur.

```
StationQuintuple quint = new StationQuintuple();
quint.departureTime = connection.departureTime;
quint.arrivalTime = Tmin;

// Additional data for journey extraction
quint.departureConnection = connection;
quint.arrivalConnection = T.get(connection.trip).arrivalConnection;
quint.transfers = numberOfTransfers;

if (S.containsKey(connection.departureStationUri)) {
  int numberOfPairs = S.get(connection.departureStationUri).size();
  StationQuintuple existingquint =
  ↪  S.get(connection.departureStationUri).get(numberOfPairs - 1);
  if (quint.arrivalTime.isBefore(existingquint.arrivalTime)) {
    if (quint.departureTime.isEqual(existingquint.departureTime)) {
      S.get(connection.departureStationUri).remove(numberOfPairs - 1);
      S.get(connection.departureStationUri).add(numberOfPairs - 1, quint);
    } else {
      S.get(connection.departureStationUri).add(quint);
    }
  }
} else {
  S.put(connection.departureStationUri, new ArrayList<StationQuintuple>());
  S.get(connection.departureStationUri).add(quint);
}
```

Code 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 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 12: Vinden van volgende vertrek bij tussenstop

2.2.2 Vertrekken en aankomsten per station

Om de zogenoemde *liveboards* te berekenen, gebruiken we een eenvoudig algoritme. 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 efficientie $O(n)$ is.

2.2.3 Route van een voertuig

De route van een voertuig af te leiden uit Linked Connections data is relatief 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 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 efficientie $O(n)$ is.

2.3 Implementatie in HyperRail

De in sectie 2.2 besproken algoritmes zijn zowel server-side in PHP, 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*[7] implementeert. Deze interface stelt de nodige

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

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[8].

Het laden van voorgaande of volgende resultaten is een belangrijke methode. De applicatie zal alle resultaten voorstellen in een lijst die verlengt wordt zodra de gebruiker het einde 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 gecached zijn voor de verzoeken die er kort op volgen.

2.3.1 Bepalen van de parameters voor server-side implementatie

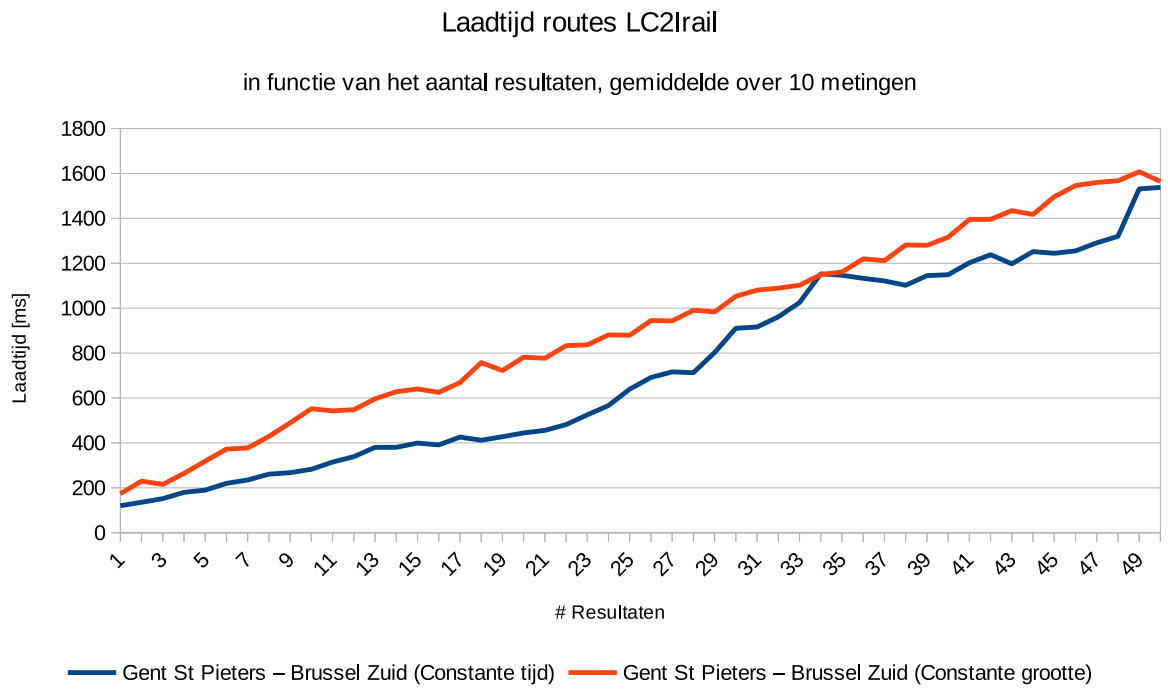
De implementatie die waarbij algoritmes op de server toepast worden, zal steeds een bepaald aantal resultaten proberen laden, om zo de overhead die HTTP requests met zich mee brengen te beperken. Hierbij moet er getracht worden om een evenwicht te vinden tussen een lange wachttijd, en het te vaak moeten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

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

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

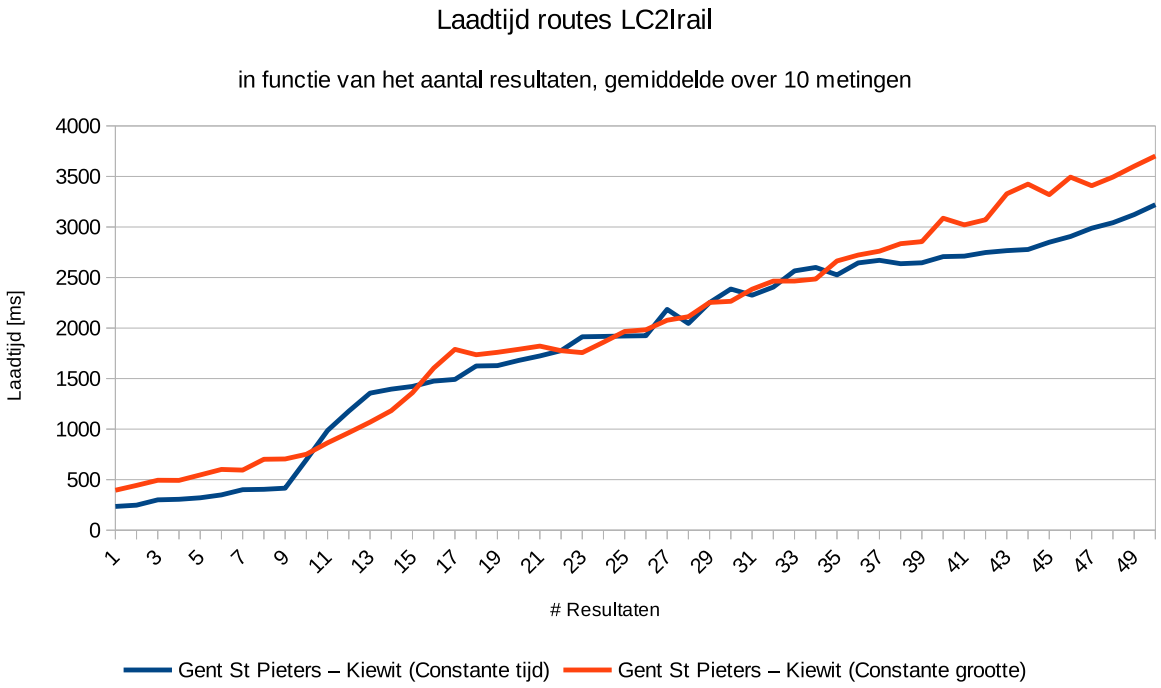
gebruikt worden, vallen deze nadelen weg: elke pagina bevat ongeveer evenveel resultaten. Het verschil tussen deze twee varianten is duidelijk zichtbaar in grafieken 2.1 en 2.2.

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 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 gepland wordt. 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.



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

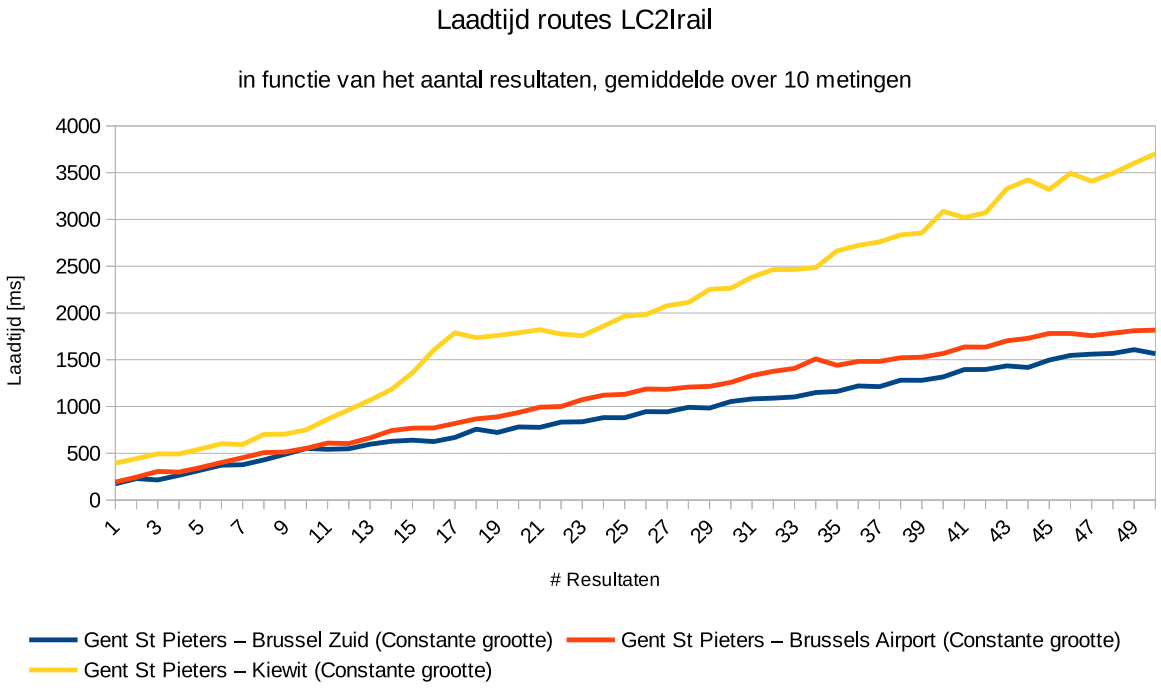
Liveboards, die informatie geven over vertrekkende en aankomende treinen in een station, bouwen we op voor een bepaald interval. Gebruikers steeds informatie willen over treinen in de komende periode, en niet enkel de eerste resultaten, gezien ze niet weten de hoeveelste trein ze zoeken. Ook voor liveboards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



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

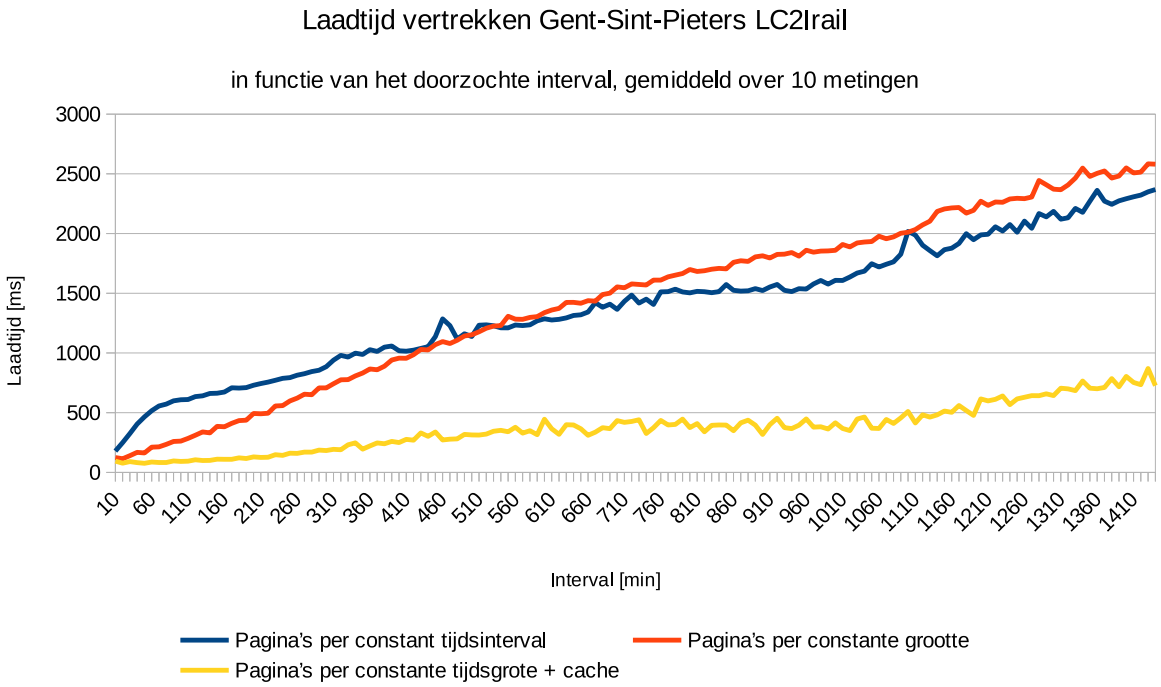
In grafiek 2.4 zien we de tijd die benodigd is om een liveboard te genereren over een bepaald interval. In deze grafieken zien we dat bij pagina's van constante tijdintervallen de responstijd initiëel sterk stijgt, waarna de curve een lineaire vorm aanneemt. Knikken in de curve kunnen we opnieuw wijten 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 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.

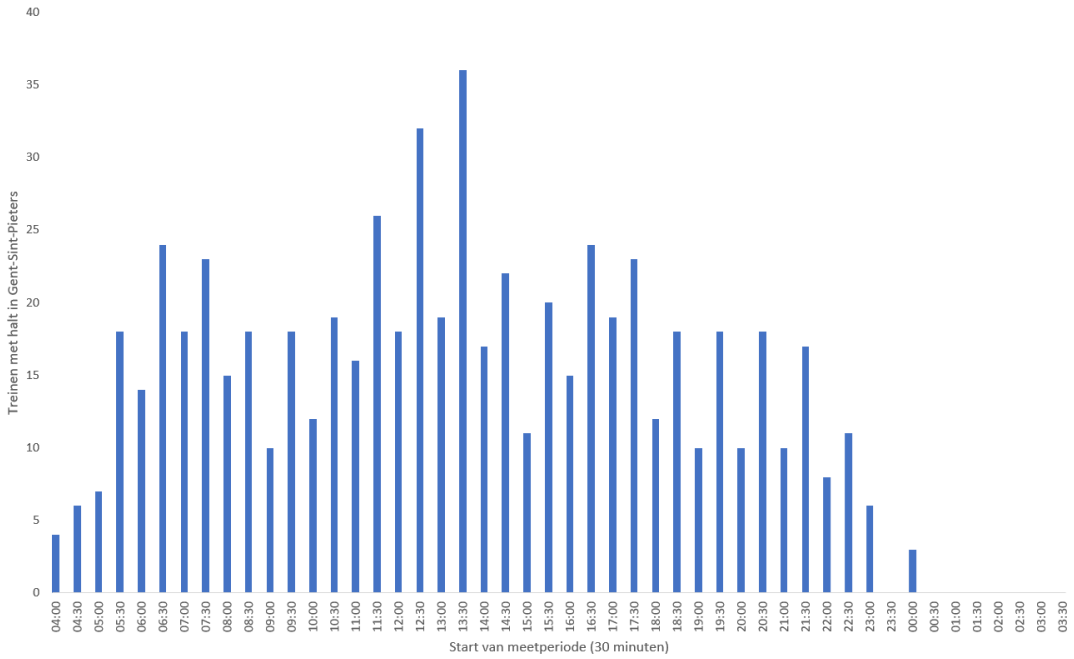


Figuur 2.3: De tijd die nodig is om routes te laden, met aankomst om 18u, in functie van het gewenste aantal resultaten. We zien een knik wanneer Linked Connections voor de nacht opgehaald worden, terwijl deze relatief weinig data bevatten.

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 dat stopt in Gent Sint Pieters op een werkdag.

2.3.2 Caching

Een van de grootste voordelen die bij Linked Connections hoort is de hoge mate waarin data gecached 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. Verder laat de keuze voor SQLite ook toe om eenvoudig te zoeken naar de pagina die informatie bevat voor een bepaald tijdstip. Een voorbeeld hiervan is te zien in fragment 13.

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

Code 13: SQLite query om juiste pagina in cache te zoeken

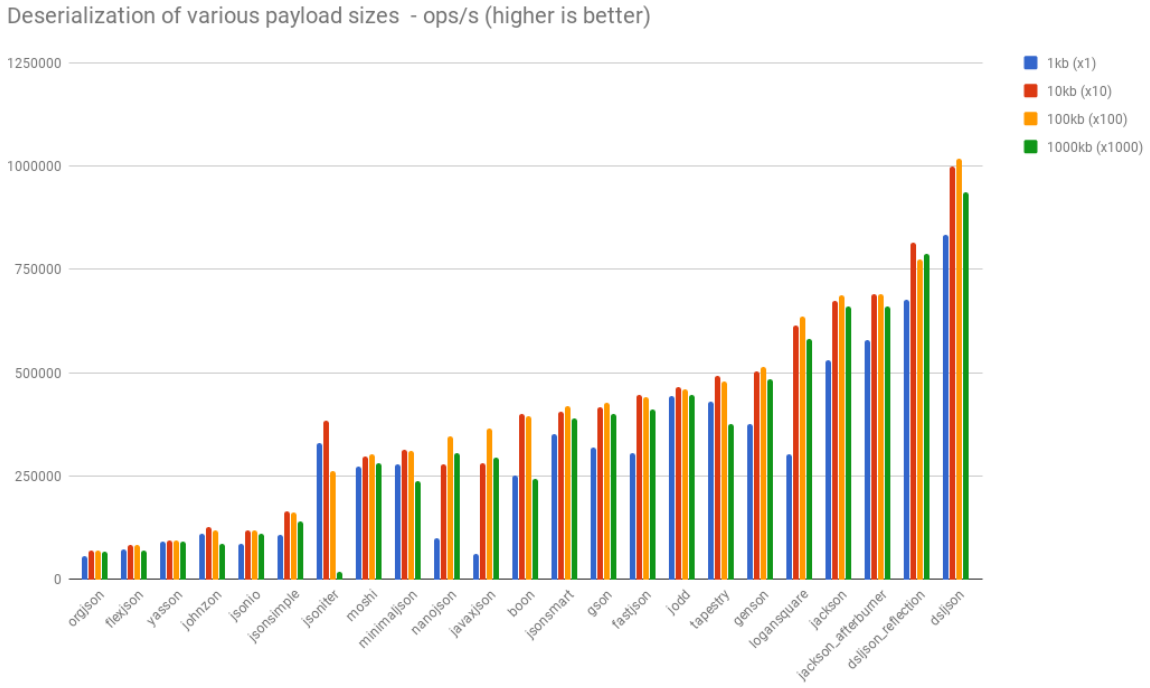
Deze cache wordt gecontroleerd alvorens een verzoek te maken. Indien geen internet beschikbaar is of de gecachte data minder dan 60 seconden oud is wordt deze hergebruikt. Deze maximale ouderdom kan ook uit de HTTP headers van de serverantwoorden onttrokken worden. De HTTP antwoord headers zijn echter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

2.4 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 geparsed, 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 duidt al snel aan dat er veel tijd besteed wordt aan het parsen van JSON, en ook specifiek aan het parsen van datums[9]. 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.

[9]Analyse van het CPU gebruik is niet precies, maar geeft een indicatie welke methodes het meeste tijd nodig hebben. Deze beperking wordt later verder toegelicht.

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 geparsed: de ongeparsede JSON wordt opgeslagen in het LinkedConnection object, en velden worden slechts geladen wanneer nodig. Dit wilt 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 verbeteringen, 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[10]. Dit is duidelijk te zien in figuur 2.6.



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

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

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

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.

"Inspirational quote"

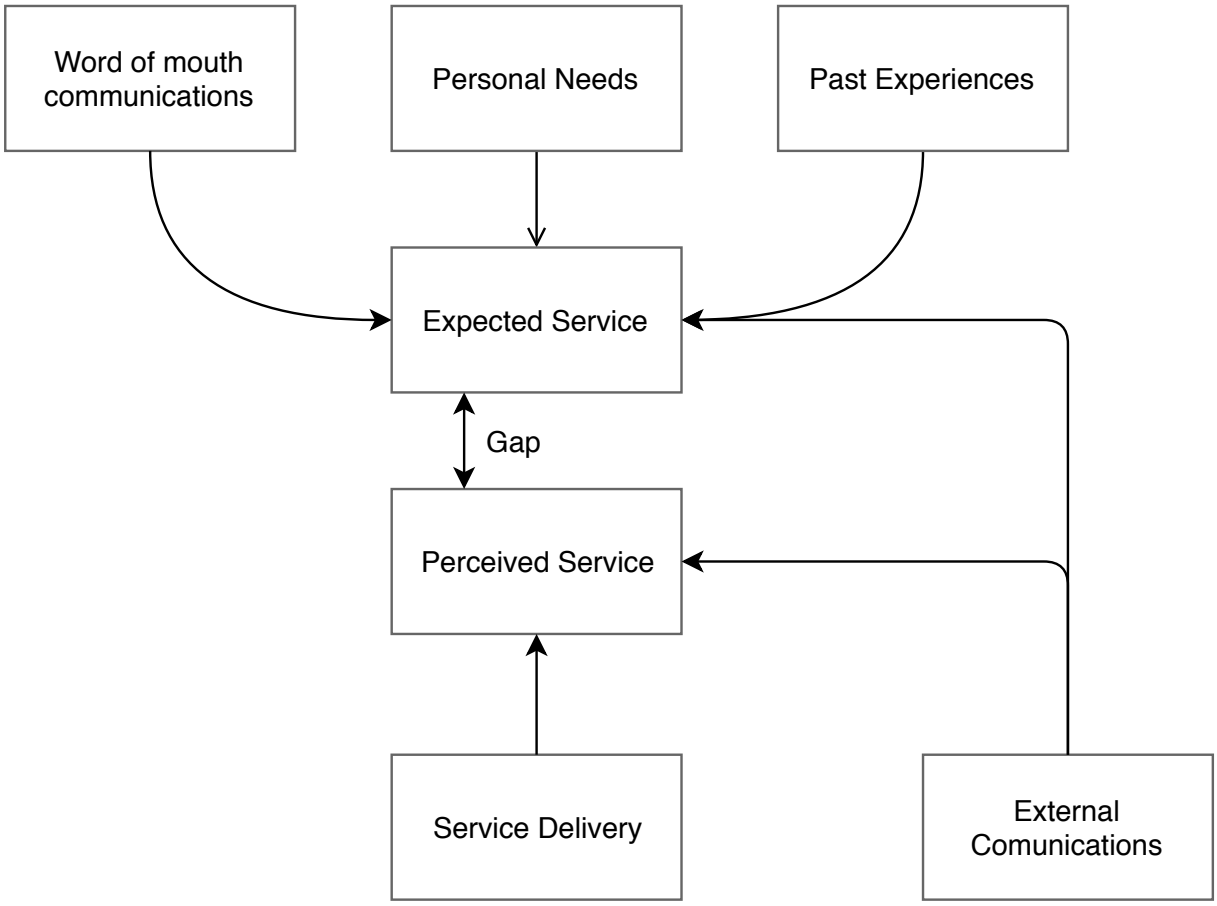
~Source

3

Onderzoek

Zoals eerder aangehaald kan de user-perceived performance afwijken van de werkelijke performance. Dit verschil wordt voornamelijk bepaald door de interface van de applicatie. Zo ervaren gebruiker 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.

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 serivce en de ervaren service, merken we ook een verschil tussen de verwachtte 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 wees uit dat interface design, prestaties van de applicatie, batterijgebruik, kostprijs van de applicatie en connectiviteit, gebruiksersroutines en levensttijl invloed hebben op de user experience [10]. In dit onderzoek zullen we ons specifiek richten op de prestaties, batterijverbruik, dataverbruik en connectiviteit.



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, 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 normaalgezien 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 het geteste station of route 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 is[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 het volledig resultaat
- de gemiddelde hoeveelheid data die verzonden en ontvangen wordt
- het gemiddeld processorgebruik van het toestel
- het gemiddeld batterijgebruik van het toestel

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:

- Bied offline informatie een meerwaarde voor gebruikers?

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

- 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.

"Inspirational quote"

~Source

4

Resultaten

Om te voorkomen dat de keuze van het geteste station of route de objectieve metingen vertekent, werden de opzoekingen van echte gebruikers gebruikt op 2 mei 2018[1]. Om de duur van de testen enigszins binnen de perken te houden, worden niet alle opzoekingen getest, maar telkens een willekeurig gekozen selectie. Deze selectie werd gemaakt door uit de chronologische logboeken met zoekopdrachten telkens de n-de zoekopdracht te gebruiken.

Voor de alle tests werd gebruik gemaakt van een HTC 10 of HTC One, verbonden met internet via wifi (ping 26ms, downloadsnelheid 42mbps, uploadsnelheid 9mbps), tenzij anders vermeld. 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.

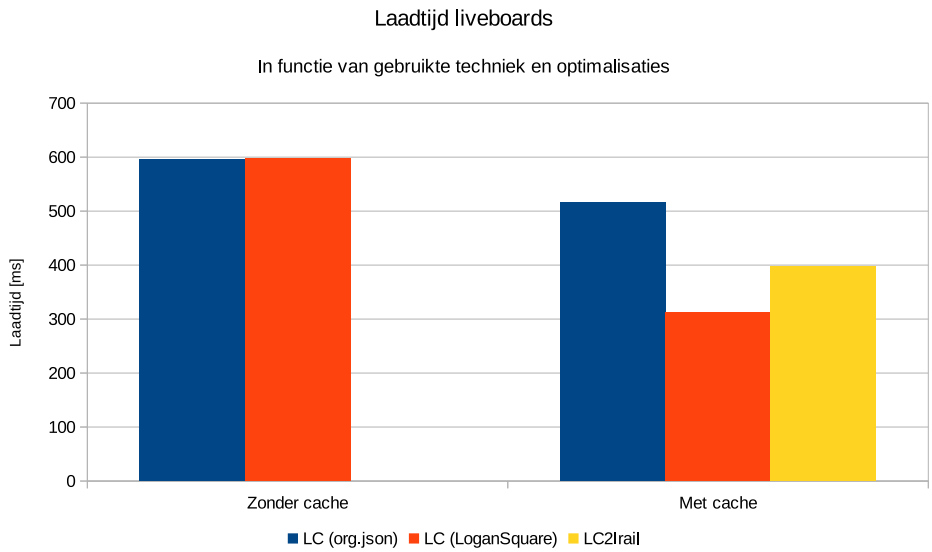
De HTC 10 is een *flagship* smartphone, uitgebracht in maart 2016. Deze beschikt over een krachtige Snapdragon 820 processor en 4GB ram. De HTC One daarintegen is een *flagship* smartphone uit 2013, welke over een oude Snapdragon 800 processor en 2GB ram beschikt. Het HTC 10 toestel komt overeen met moderne toestel bovenaan in het mid-range aanbod. Dit toestel biedt goede prestaties en zullen we gebruiken om de performantie van Linked Connections op toestellen van 400 a 600 euro te testen. De HTC One komt op vlak van prestaties overeen met low-end toestellen en oudere mid-range toestellen, en komt ongeveer overeen met de prestaties

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

van een paar jaar oude smartphone van 200 euro. Op deze manier krijgen we een indicatie van de performantie voor alle smartphonegebruikers.

4.1 Liveboards

4.1.1 Metingen



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

Variant	parser	cache	minimaal (ms)	gemiddelde (ms)	maximaal (ms)
LC op toestel	org.json	nee	302	595	3471
LC op toestel	org.json	ja	409	516	3599
LC op toestel	LoganSquare	nee	392	597	2027
LC op toestel	LoganSquare	ja	166	313	3428
LC op server			232	397	1421

Tabel 4.1: De gemeten laadtijd voor het eerste resultaat liveboards gebruikmakend van een HTC 10 voor 262 opzoekingen gebaseerd op de iRail logs.

In tabel 4.1 en grafiek 4.1 zijn de 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 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 implementatiedetails van dezelfde techniek valt wegens tijdsgebrek buiten het bereik van deze masterproef.

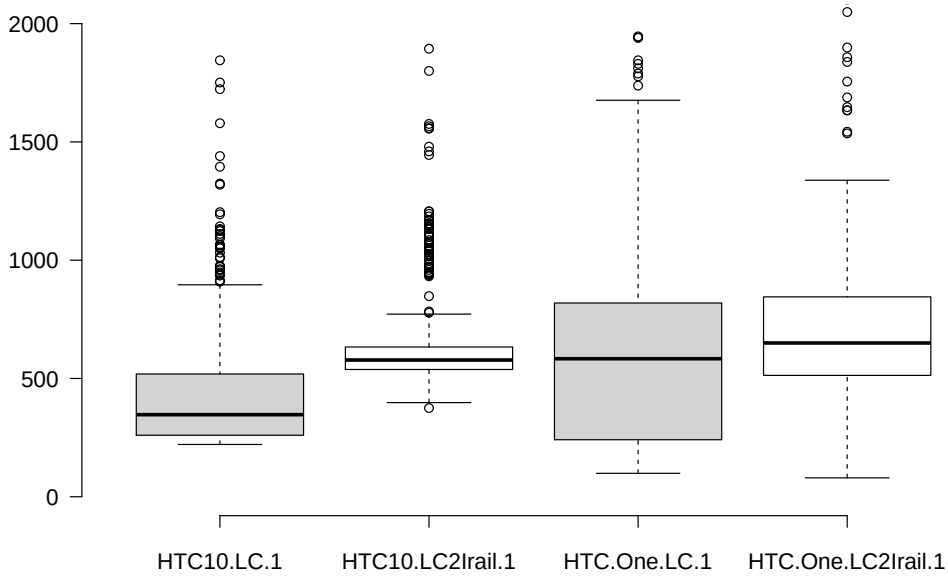
In deze cijfers is invloed van de cache wel duidelijk merkbaar. We zien wel een duidelijk verschil tussen de parsers: terwijl bij gebruik van de *LoganSquare* parser de gemiddelde laadtijd bijna halveert, terwijl het effect van de cache bij het gebruik van de *org.json* parser veel kleiner is. Wanneer de cache uitgeschakeld is is het verschil tussen de parsers verwaarloosbaar. Dit is te verklaren door het feit dat voor het tonen van vertrekken of aankomsten 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 telkens de vijfde opzoeking uit de iRail logs. Voor elk liveboard worden twintig resultaten geladen. De resultaten hiervan zijn zichtbaar in grafieken 4.4, 4.5 en 4.6, 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 lokale implementatie deze steeds zal herberekenen vanaf Linked Connections pagina's.
  - 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 is in praktijk amper merkbaar voor gebruikers
- Wanneer we naar de mediane performantie kijken, is Linked Connections duidelijk sneller voor de eerste resultaten. Dit snelheidsverschil is aanzienlijk, en vooral te wijten aan het feit dat Linked Connections volledige ondersteuning biedt voor incrementele resultaten, waarbij de server steeds meerdere pagina's zal overlopen voor een antwoord gegeven wordt; Naarmate meer resultaten geladen worden, vormt het overlopen van meerdere pagina's een voordeel voor de serverimplementatie: Zo zullen in daluren sneller resultaten geladen worden door de snellere toegang tot Linked Connections pagina's, terwijl bij Linked Connections sprake is van een *overhead* door het laden van te veel data. Hierbij zien we dat hoe sneller het toestel, hoe langer Linked Connections het snelst blijft. Ook blijft het

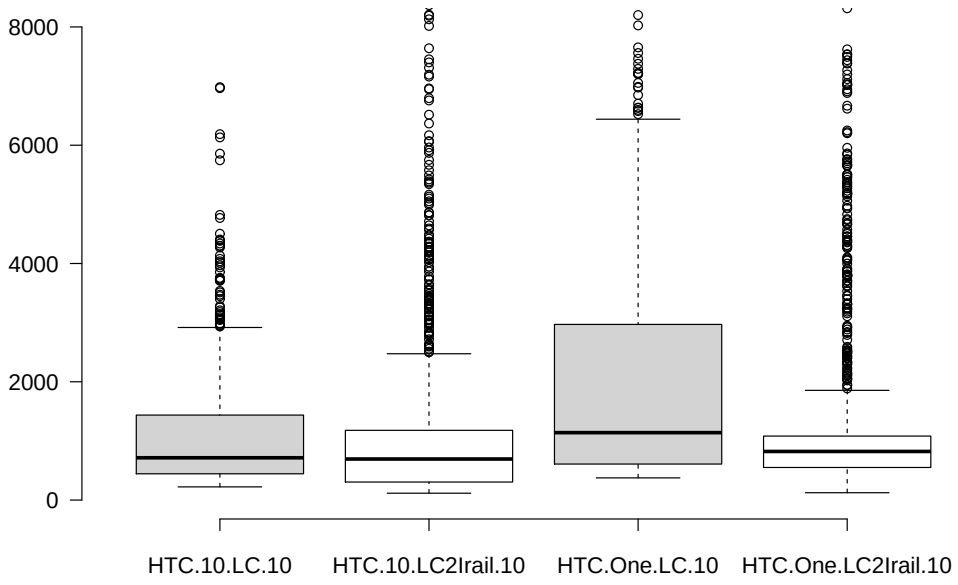
verschil tussen Linked Connections en LC2Irail beperkt op de HTC 10, terwijl dit verschil aanzienlijk oploopt op de HTC One.

- 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 verklaart 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.



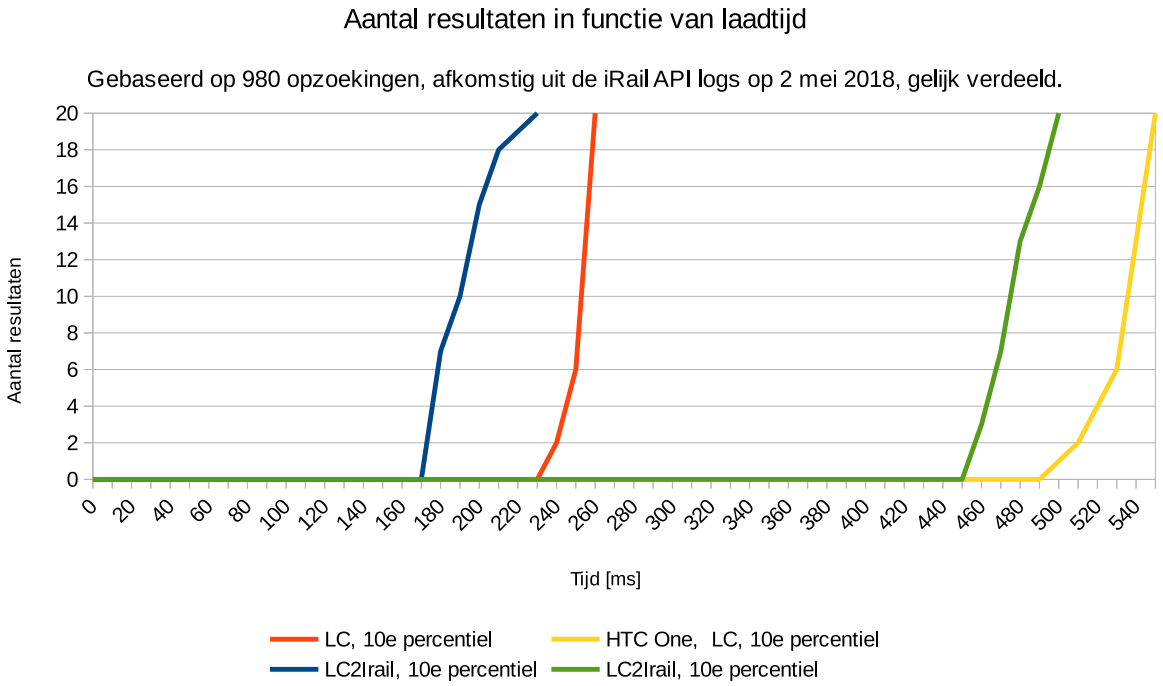
Figuur 4.2: Laadtijd eerste resultaat liveboard in functie van toestel en technologie.

Waneer we nu naar de spreiding van de laadtijden kijken, zichtbaar in figuren 4.2 en 4.3 zien we ook hier duidelijke verschillen tussen de verschillende methodes. 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 duidt. 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

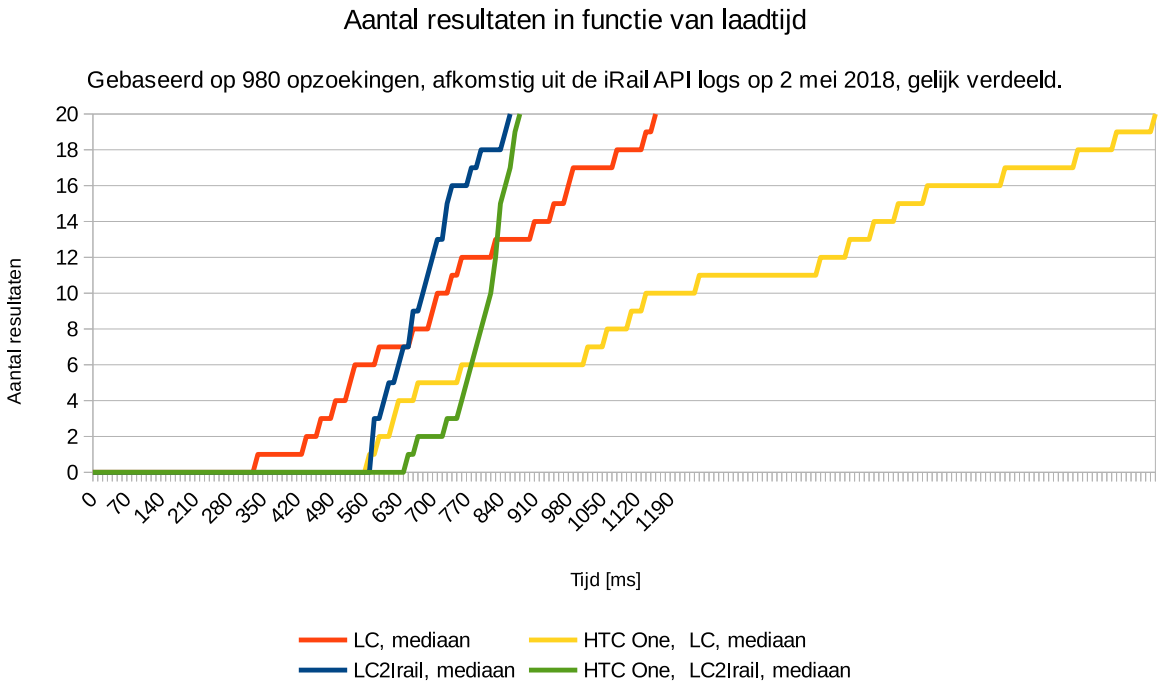


Figuur 4.3: Laadtijd tiende resultaat liveboard in functie van toestel en technologie.

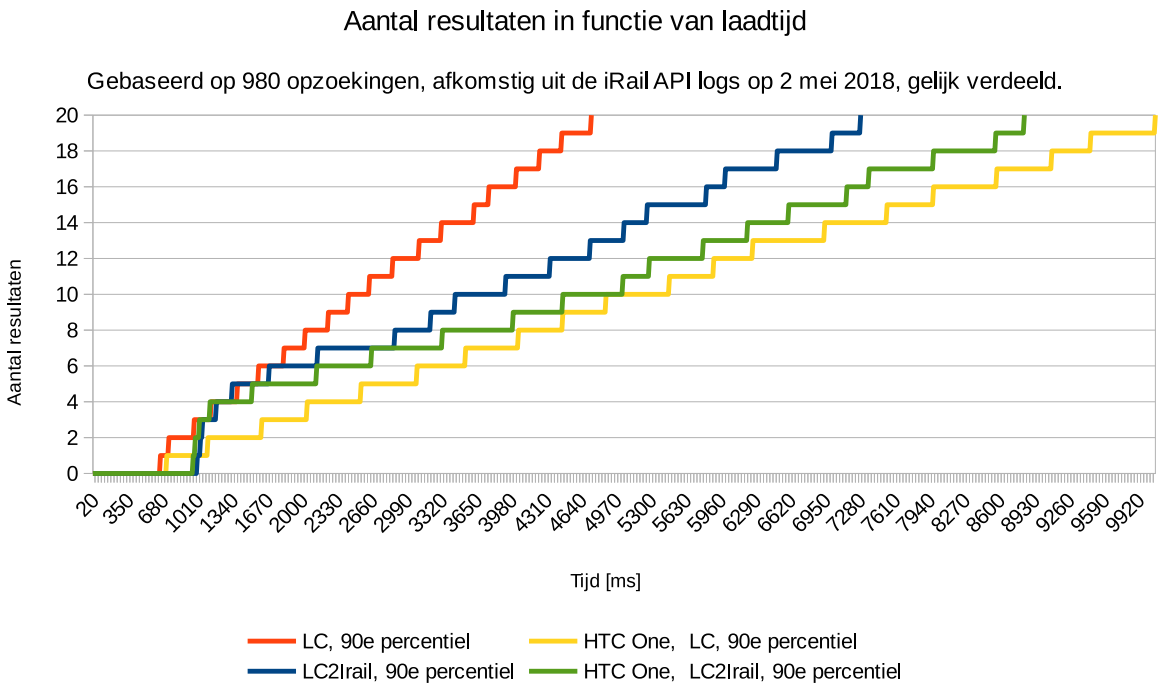
is een ouder toestel niet enkel trager, maar is ook de spreiding veel groter, en zijn de resultaten dus minder consistent op oudere (tragere) toestellen.



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



Figuur 4.5: Het aantal resultaten in functie van de verlopen tijd.

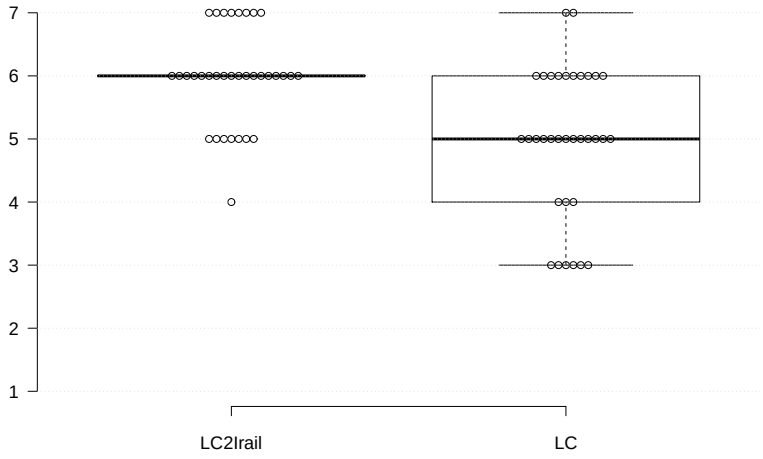


Figuur 4.6: Het aantal resultaten in functie van de verlopen tijd.

4.1.2 Ervaringen

Wanneer we nu naar de ervaringen van gebruikers gaan kijken, stemmen deze ongeveer overeen met wat we zouden verwachten na het bekijken van voorgaande grafieken.

Zoals we in figuren 4.2 en 4.3 konden zien blijkt uit testen dat de performantie van LC2Irail consistenter is. Ook bij de gebruikerservaring zien we dit terugkomen. Wanneer de ervaren snelheid wordt uitgezet in een boxplot per implementatie, 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.



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.

Hoewel 12 van de 17 testpersonen aangeeft licht tot extreem tevreden te zijn met de laadsnelheid, geven slechts 2 personen aan dat de lijst met vertrekken sneller laadt bij de lokale implementatie van Linked Connections. Nog eens 3 personen geven aan beide implementaties even snel te ervaren.

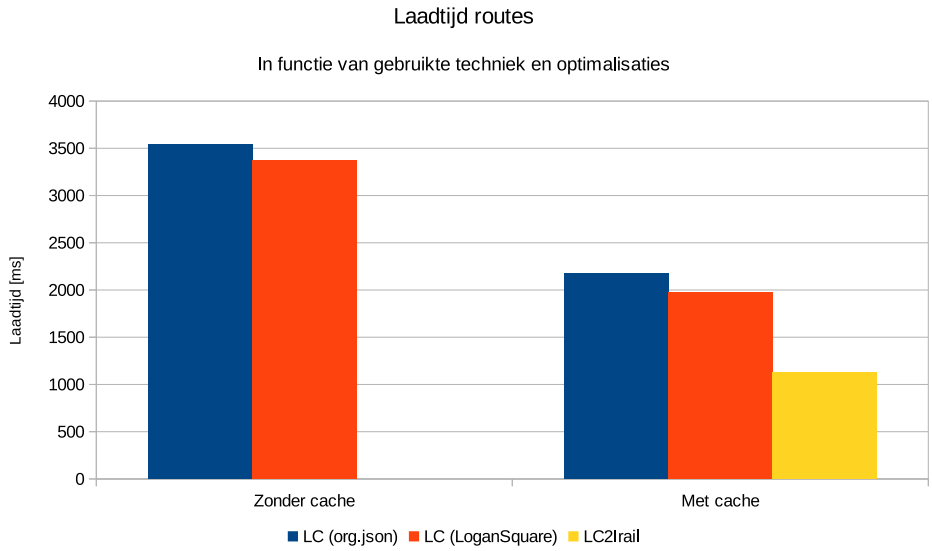
Wanneer we gaan kijken naar de verschillen tussen de JSON parsers, blijkt dat beide parsers ongeveer even goed presteren in de ogen van de gebruikers. Respectievelijk 5 op 7 en 7 op 10 gebruikers zijn neutraal of tevreden, en bij beide varianten is er telkens een gebruiker neutraal. Deze vergelijking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

Wanneer we kijken 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.

Tot slot werden alle testpersonen ook expliciet gevraagd welke implementatie ze als sneller ervaarden. Hierbij waren de antwoorden verdeeld: acht personen kozen de eerste variant, vier personen hadden geen mening, en vijf personen kozen de tweede variant. Hierbij valt op dat bij de eerste lokale implementatie,

4.2 Routes

4.2.1 Metingen



Figuur 4.8: De gemeten laadtijd voor routes gebruikmakend van een HTC 10 voor 779 opzoekingen gebaseerd op de iRail logs.

In tabel 4.2 en grafiek 4.8 zijn de 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

Variant	parser	cache	minimaal (ms)	gemiddelde (ms)	maximaal (ms)
LC op toestel	org.json	nee	401	3539	8531
LC op toestel	org.json	ja	221	2172	6960
LC op toestel	LoganSquare	nee	386	3374	7554
LC op toestel	LoganSquare	ja	233	1973	7640
LC op server			27	1126	3374

Tabel 4.2: De gemeten laadtijd voor routes gebruikmakend van een HTC 10 voor 779 opzoekingen gebaseerd op de iRail logs.

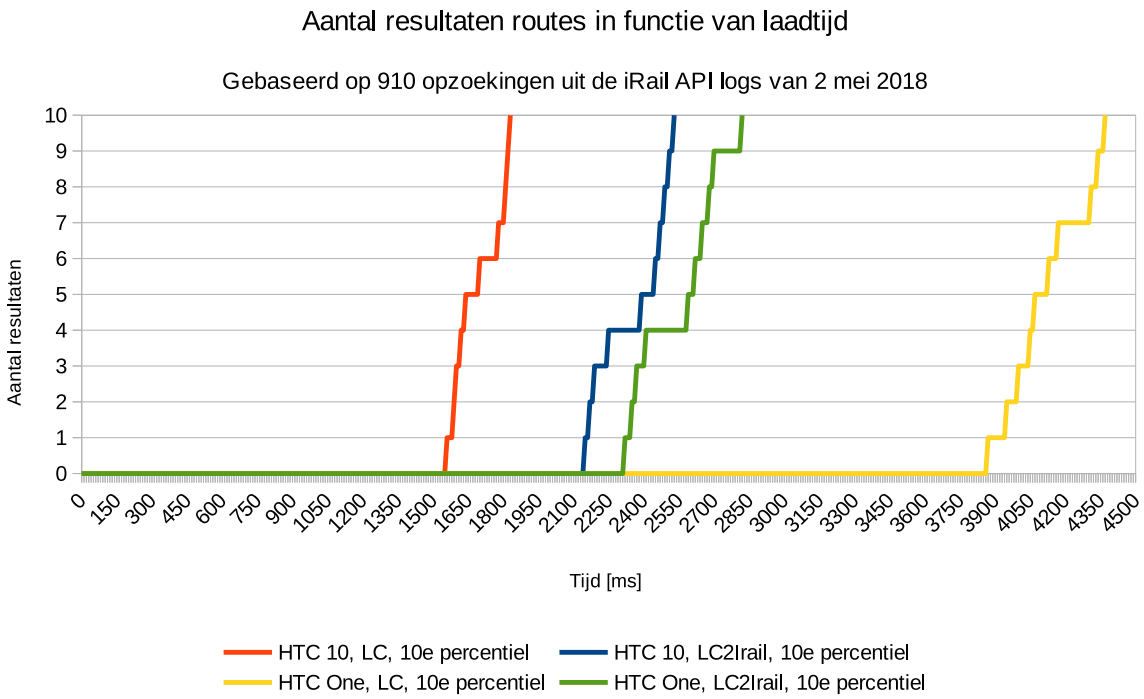
van de snelheid - een volledige en diepgaande statistische analyse van de performantieverschillen tussen verschillende implementatiedetails van dezelfde techniek valt wegens tijdsgebrek buiten het bereik van deze masterproef.

Net zoals bij liveboards is ook hier de invloed van de cache duidelijk merkbaar. Vergeleken met dezelfde analyse voor Liveboards (figuur 4.1), zien we hier een minder groot verschil tussen de parsers: er moeten grote hoeveelheden data verwerkt worden, en het algoritme om de data 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 geladen. De resultaten hiervan zijn zichtbaar in grafieken 4.9, 4.10 en 4.11, respectievelijk voor het tiende, vijftigste en negentigste percentiel.

Uit deze grafieken kunnen we opnieuw duidelijke verschillen zien:

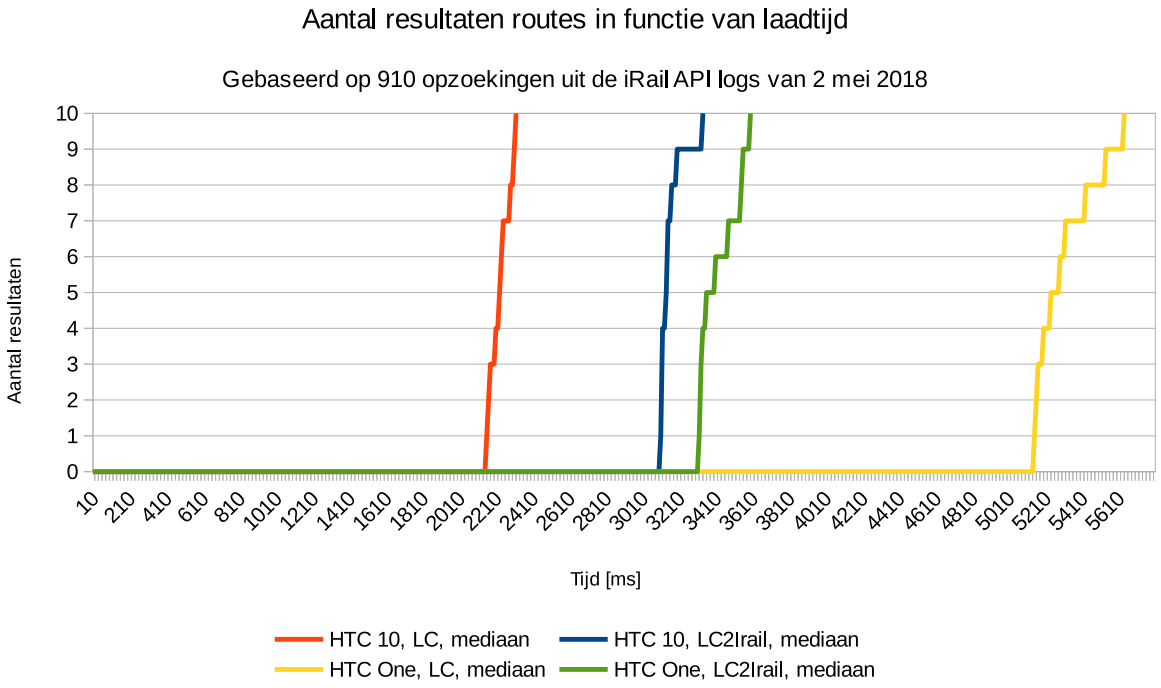
- In alle grafieken en voor alle testen, hebben de curves een gelijkaardige vorm, waarbij er na een relatief lange wachttijd aan snel tempo resultaten geladen worden: in het geval van Linked Connections is voor de eerste opzoeking telkens een relatief grote hoeveelheid data nodig is, waarna slechts één 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 zit. In het geval van LC2Irail worden resultaten ook onmiddelijk voor grote intervals opgehaald, om zo het aantal verzoeken te beperken.
- Terwijl in het alle gevallen Linked Connections beter presteert op de HTC 10, presteert het slechter op de HTC One.
- Opnieuw presteert LC2Irail op beide toestellen gelijkaardig, met slechts een kleine verschuiving in tijd tussen beide curves.
- Terwijl in het slechtste geval bijna alle varianten gelijk presteren, loopt Linked Connections



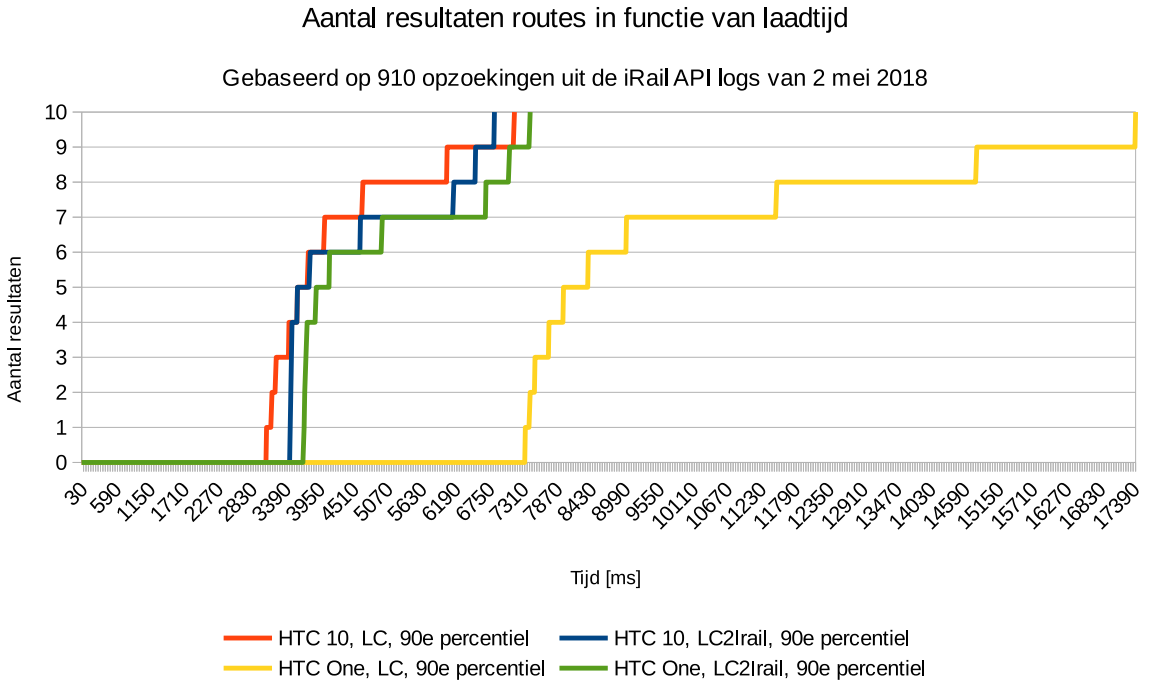
Figuur 4.9: Het aantal resultaten in functie van de verlopen tijd.

op de HTC 10 enorm achter.

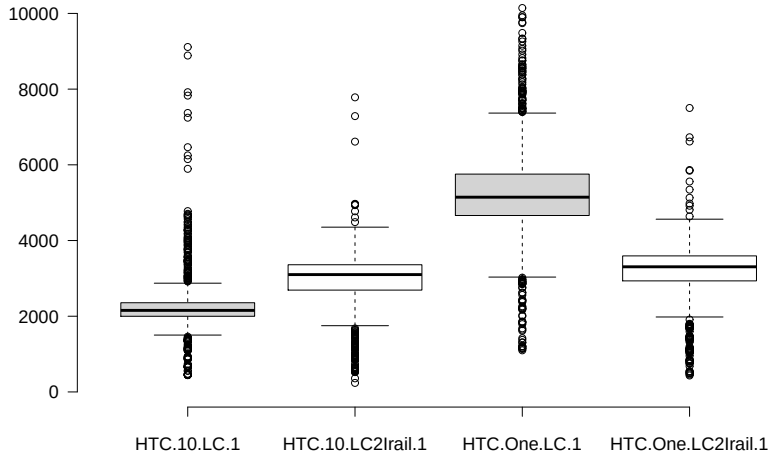
Wanneer we nu specifiek naar de verdelingen kijken, gevisualiseerd door middel van box plots in figuur 4.12 en 4.13, Zien we ook hier duidelijk hoe LC2Irail gelijke prestaties heeft op beide toestellen, terwijl de prestaties van Linked Connections sterk variëren per toestel. Op de HTC 10 zal al meer dan 75% van de opzoekingen geladen zijn op het moment dat LC2Irail op hetzelfde toestel minder dan 25% van de verzoeken beantwoordt heeft. Op het HTC One toestel is dit echter omgekeerd, en nog extremer.



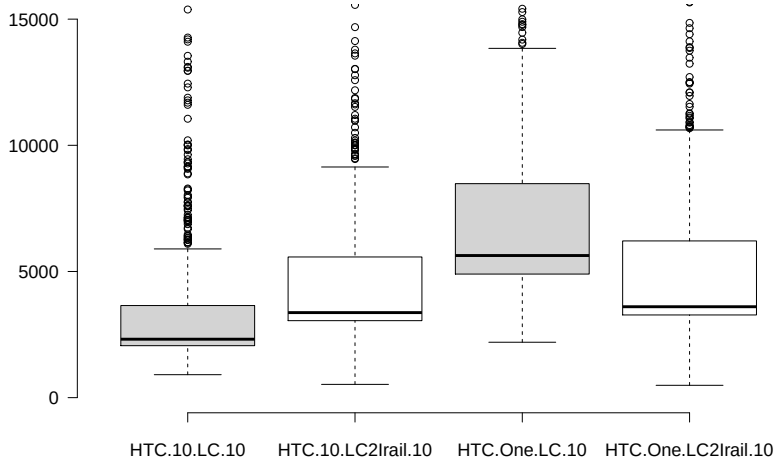
Figuur 4.10: Het aantal resultaten in functie van de verlopen tijd.



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



Figuur 4.12: Laadtijd eerste resultaat route in functie van toestel en technologie.



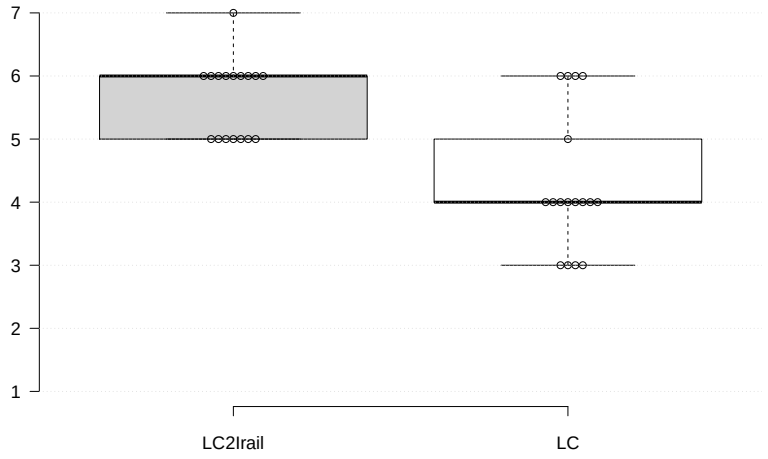
Figuur 4.13: Laadtijd tiende resultaat route in functie van toestel en technologie.

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 snelheid voor beide implementaties ongeveer gelijk ervaren wordt.

Wanneer we nu de resultaten van user testing vergelijken met de verwachtingen, blijken deze verwachtingen grotendeels in vervulling te gaan. In figuur 4.14 is te zien dat de prestaties van LC2Irail iets consistenter beoordeeld worden, en LC2Irail tevens een betere beoordeling krijgt dan Linked Connections.

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.



Figuur 4.14: De ervaren snelheid op een schaal 1-7 van routes voor LC2Irail en Linked Connections, gebaseerd op 17 user tests.

4.3 Voertuigen

4.3.1 Metingen

Het opzoeken van het traject dat een voertuig aflegt heeft als groot verschil dat incrementele resultaten niet door de gebruikte applicatie ondersteund worden. De reden hiervoor is dat het traject van het voertuig het enige en volledige resultaat is dat de gebruiker wenst, in tegenstelling tot liveboards en routes, waar de gebruiker niet het volledige, maar slechts een deel van het resultaat wenst te zien.

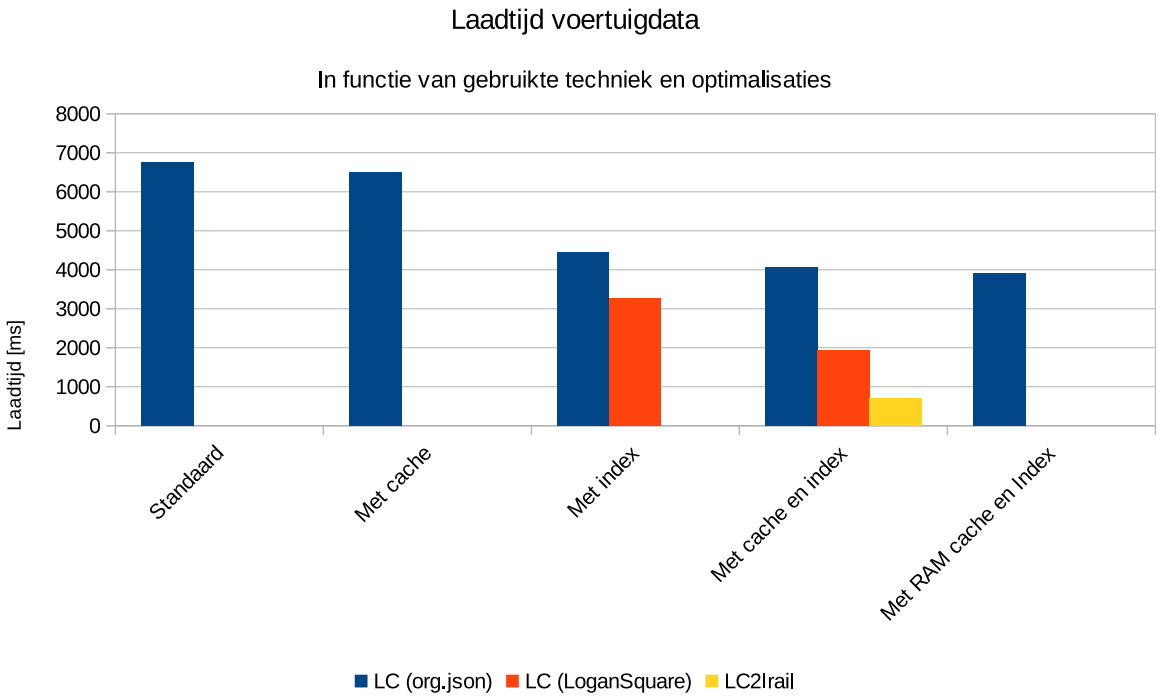
Dit is ook de opzoeking die het meeste data vereist bij Linked Connections: alle pagina's moeten doorzocht worden op connecties met betrekking tot één specifiek voertuig. Dit voertuig komt slechts in een relatief beperkt aantal pagina's voor, gezien het voertuig slechts enkele uren rijdt, en het tijdstip van eerste 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 minimale, gemiddelde en maximale opzoektijd hiervoor is te zien in tabel 4.3. De gemiddelde resultaten zijn tevens gevisualiseerd in figuur 4.15.

Variant	parser	cache	index	minimaal (ms)	gemiddelde (ms)	maximaal (ms)
LC op toestel	org.json	nee	nee	540	6764	12676
LC op toestel	org.json	ja	nee	483	6488	10921
LC op toestel	org.json	nee	ja	2638	4443	10956
LC op toestel	org.json	ja	ja	2440	4066	6003
LC op toestel	org.json	RAM	ja	2263	3912	5763
LC op toestel	LoganSquare	nee	ja	1860	3283	5374
LC op toestel	LoganSquare	ja	ja	1195	1925	2888
LC op server				264	713	5068

Tabel 4.3: De gemeten laadtijd voor voertuigen gebruikmakend van een HTC 10 voor 102 opzoekingen gebaseerd op de iRail logs.

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



Figuur 4.15: De gemeten laadtijd voor voertuigen gebruikmakend van een HTC 10 voor 102 opzoekingen gebaseerd op de iRail logs.

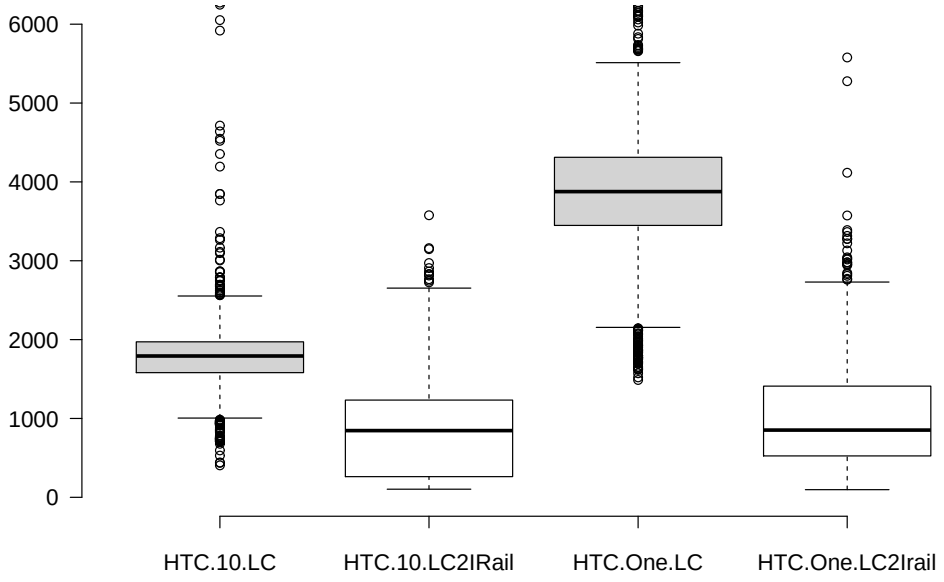
worden, is een drastische verbetering merkbaar. Het gemiddelde daalt in deze beeperkte test met ongeveer een derde. Toevoeging van een cachegeheugen, op flash of in het RAM geheugen, brengt ook hier slechts weinig beterschap.

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 1620 voertuigen op die plaatsvinden op 6 mei 2018. Dit wordt enerzijds gedaan voor de lokale implementatie die gebruik maakt van de LoganSquare parser, cache en lokale index, en anderzijds voor de serverimplementatie, die server-side over dezelfde index en een cache beschikt.

Wanneer we kijken naar de box-plot van de responstijd, weergegeven in figuur 4.16, zien we dat de lokale implementatie duidelijk slechter presteert. Op beide toestellen is LC2Irail sneller dan Linked Connections. Bij de HTC 10, een snel toestel, valt dit nog enigszins mee, maar op de

HTC One zijn de meeste resultaten binnen 3000 milliseconden geladen, terwijl op dat moment nog geen 25% van de opzoekingen via Linked Connections uitgevoerd werd. Ook zien we hier dat LC2Irail consistente prestaties biedt: beide box plots zijn praktisch 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 heeft een grotere variatie in de laadtijd.

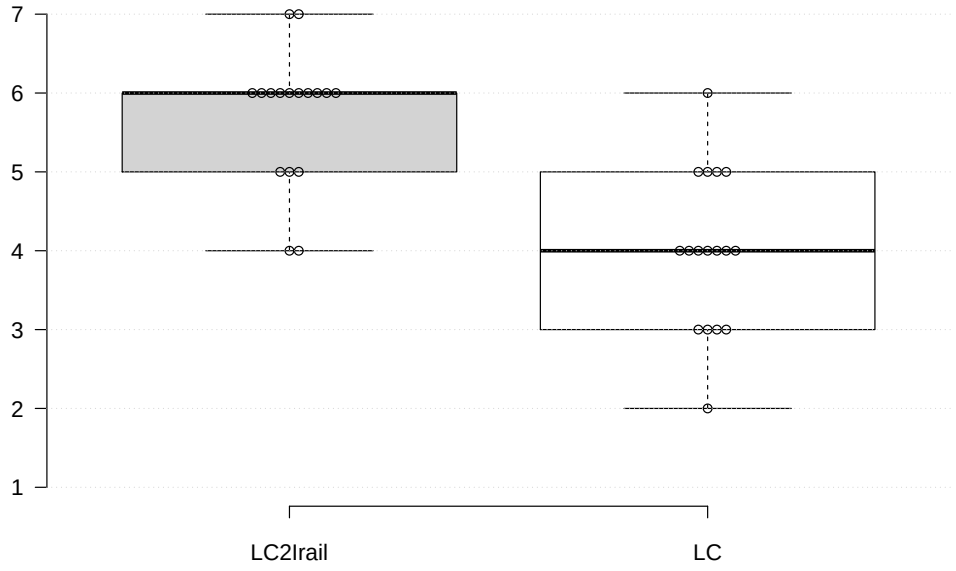


Figuur 4.16: De prestaties voor het laden van voertuigen, gemeten door alle voertuigen, beschreven in Linked Connections, voor 6 mei op te zoeken.

4.3.2 Ervaringen

Wanneer we nu 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 werd. In figuur 4.16 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

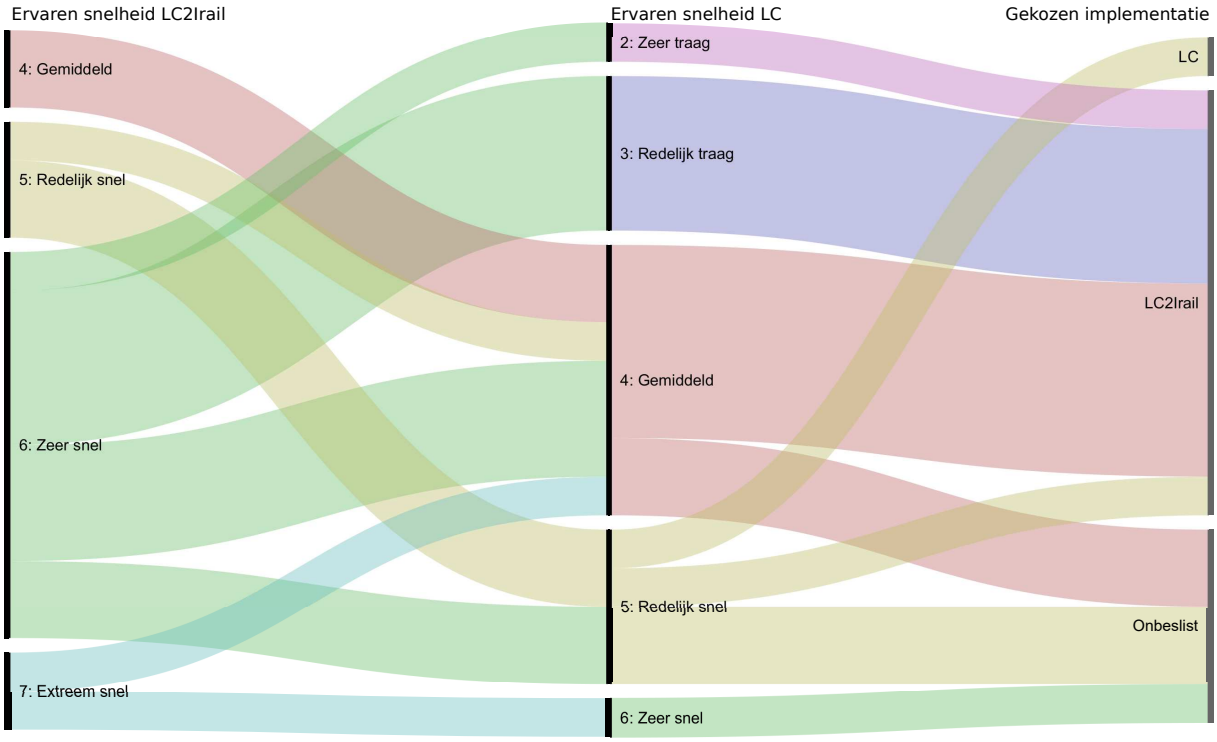
een veel grotere spreiding is in de gegeven antwoorden, terwijl bij LC2Irail iedereen het er over eens lijkt dat deze implementatie snel is.



Figuur 4.17: De ervaren snelheid op een schaal 1-7 van routes voor LC2Irail en Linked Connections, gebaseerd op 17 user tests.

Personen die de lokale implementatie op basis van *LoganSquare* testten, beoordeelden de laadtijd iets beter vergeleken met de groep die de implementatie op basis van de *org.json* parser testte. Ondanks dat de testgroep onvoldoende groot was om een veralgemening te kunnen maken, kunnen we 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 4.18 is duidelijk te zien hoe de ervaringen van elke gebruiker waren. Zo zien we dat de ervaring voor gebruikers nooit verbeterd, en veel gebruikers een groot verschil ervaren tussen de snelheid van



Figuur 4.18: Verbanden tussen de door gebruikers gekozen implementaties voor voertuigen.

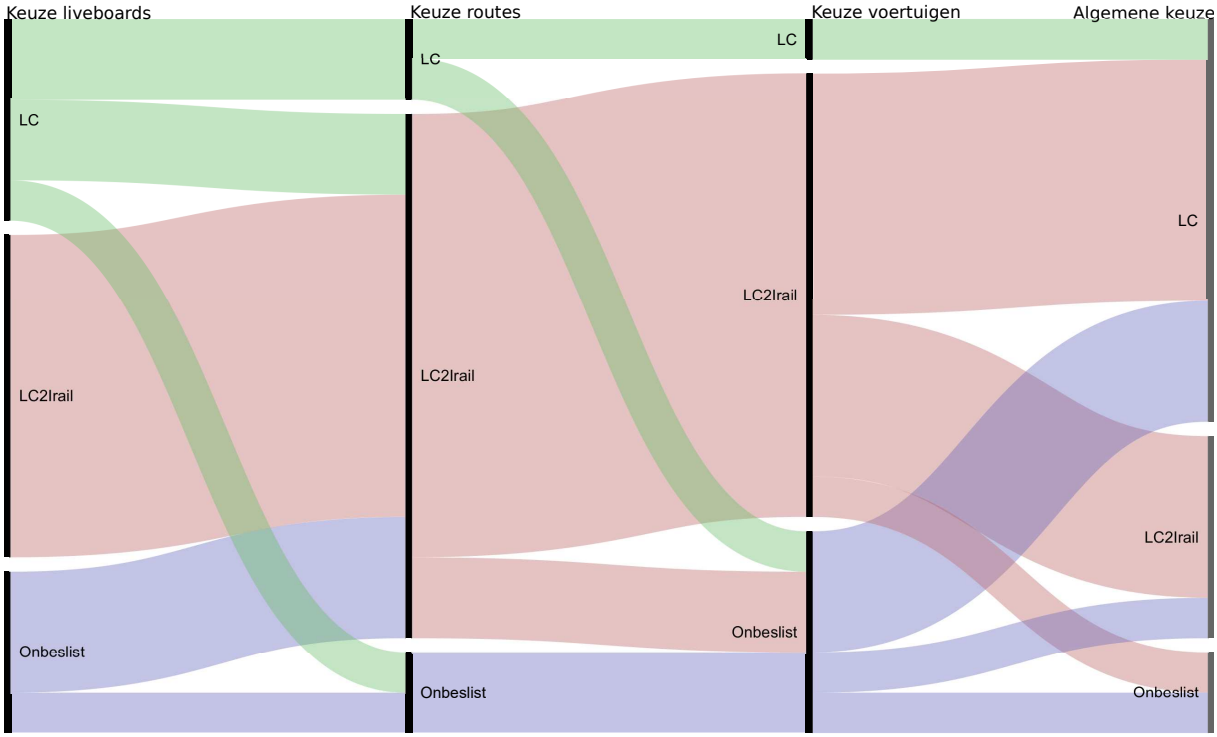
beide implementaties. Veel gebruikers die Linked Connections als redelijk of zeer snel ervaren, ervoeren LC2Irail nog steeds als sneller, waardoor ze wanneer ze moesten kiezen niet voor Linked Connections kozen.

Uit de combinatie van metingen en ervaringen concluderen we dat gebruikers de laadtijd van voertuigdata acceptabel vinden wanneer deze onder 1 seconde blijft. Een seconde langer laden, wat vaak geval is bij Linked Connections, wordt door de meeste personen niet meer als snel ervaren. Gebruikers zullen in dit geval de voorkeur geven aan een snellere applicatie.

4.4 Keuze van de gebruiker

Voor alle soorten informatie (liveboards, routes, en voertuigen) lijkt Linked Connections een slechtere gebruikerservaring 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 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.

Dit zien we ook terug in de antwoorden wanneer gebruikers gevraagd werd te kiezen tussen beide implementaties. Voor vertrekken zijn er vier gebruikers die beide implementaties even snel vinden, terwijl van de overige 13 slechts vijf kiezen voor Linked Connections. Bij routes en voertuigen scoort Linked Connections zoals verwacht slechter: er zijn respectievelijk slechts twee en een gebruiker die voor Linked Connections kiezen zijn. Hierbij zijn er respectievelijk twee en vijf gebruikers die geen verschil tussen de implementaties merkten. De keuzes van gebruikers werden uitgezet in figuur 4.19. In dit diagram zien we zowel hoeveel gebruikers voor elke implementatie kozen, maar ook hoe de keuze van gebruiker 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.



Figuur 4.19: Verbanden tussen de door gebruikers gekozen implementaties voor alle soorten informatie, alsook de resulterende keuze waarbij ook offline toegang in rekening werd gebracht.

Terwijl de meerderheid van de gebruikers steeds voor LC2Irail koos, kantelt deze balans echter volledig om wanneer gebruikers worden gevraagd om met alle aspecten rekening te houden. Dit is duidelijk zichtbaar aan de rechterkant van figuur 4.19. Hieruit blijkt duidelijk dat gebruikers enige snelheid willen opgeven in ruil voor offline opzoekingen. Zeven gebruikers laten weten dat ze een hybride systeem ideaal zou zijn, waarbij de snelheid van LC2Irail gecombineerd wordt met Linked 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.

4.5 Beperkingen

4.5.1 Kleine steekproef voor user testing

Zoals eerder vermeld ontbreekt op het moment van schrijven nog cruciale informatie in Linked Connections, 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.

4.5.2 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.

4.5.3 Processorverbruik niet meetbaar

De Android CPU Profiler beïnvloed 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[2]. Deze problemen treden op door de nieuwe Android CPU profiler, die zelf teveel processortijd op het apparaat vereist.

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

4.5.4 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.

"Inspirational quote"

~Source

5

Interpretatie

We zullen nu de resultaten, besproken in hoofdstuk 4, proberen interpreteren om een antwoord te vinden op welk transportformaat het best geschikt is voor routeplanning API's, om de beste gebruikerservaring te bekomen. Het is onmiddellijk duidelijk dat het moeilijk wordt hier een eenduidig antwoord op te vinden.

Voor liveboards blijkt dat, afhankelijk van het gebruikte toestel Linked Connections concurrentieel is op vlak van snelheid. 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 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.

Verder blijkt ook dat het moeilijk is om een oorzaak van dit snelheidsverschil aan te duiden. 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 pres-

taties boodt, 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 zijn dan ook moeilijk te vatten. Echter is er een zeer belangrijke ”externe”invloed op de uitvoeringstijd, namelijk de *Java Virtual Machine* 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. Hierbij kunnen fabrikanten kiezen om de *garbage collection* agressiever in te stellen, wat leid tot efficienter geheugengebruik ten koste van prestaties. 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 netwerk verkeer 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 verzend, 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.

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 wilt zeggen dat gebruikers Linked Connections snel genoeg ervaren om een algemene betere gebruikerservaring te bieden vergeleken met RPC API’s.

"Inspirational quote"

~Source

6

Conclusie

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] 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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

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

[10] 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.

[11] 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/

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

[13] 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

[14] 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

[15] 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

[16] 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

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

[18] 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

[19] 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

[20] 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.

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?

7

Resultaten user testing