Publishing real time Open Data

By Dylan Van Assche

KU LEUVEN

BE OPEN KNOWLEDGE BELGIUM

Are you sure you're going to make it?

> Op een halfuur tijd krijgen we soms 9 verschillende versies van wat we moeten doen.

Treinbegeleider

Reizigers De Lijn klagen over stiptheid, maar deze obstakels vinden chauffeurs dagelijks op hun weg

ACOD: "Minister Weyts brengt steeds goednieuwsshow, terwijl realiteit anders is"

Minder dan de helft van treinen rijdt op tijd: stiptheid vooral in de piekuren dramatisch

IB , ADN , KG , KV en TT | 23 januari 2019 | 17u08 | Bron: Belga

Openbaar vervoer

Tevredenheid over De Lijn zakt naar nieuw dieptepunt

NMBS stelt vertragingen te rooskleurig voor, boos personeel doet boekje open: "Amateurisme ten top"

> Wanneer we de vertragingen die twintig pendelaars voor ons bijhielden naast de officiële cijfers leggen, merken we grote verschillen.

Research questions

1. How can we keep a cache for route planners up to date in a cost-efficient way for both Open Data publishers and consumers?
2. How can we consume real time Open Data updates in route planner algorithms in an efficient way?

Specifications

Implementation

Experiments

Conclusion

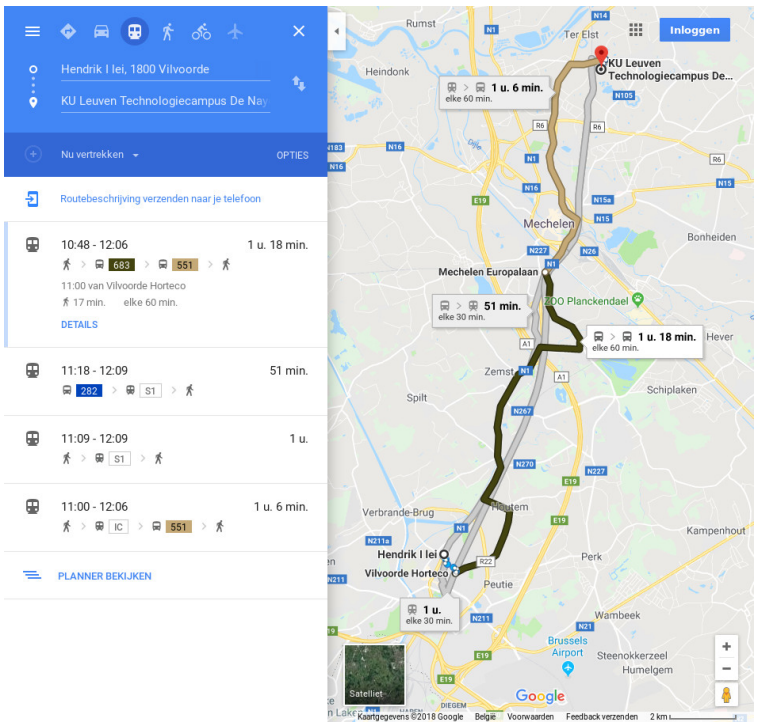
Goals

Informing commuters in real time.

Alternatives based on real time data.

Cost-efficient for publishers and consumers

Google has been doing this for years



How can we publish Linked Connections in a more efficient way?

Open Data

> *Open data is data that can be freely used, re-used and redistributed by anyone - subject only, at most, to the requirement to attribute and sharealike.*
>
> ***opendatahandbook.org***

Reducing maintenance and publication costs

Specifications

Implementation

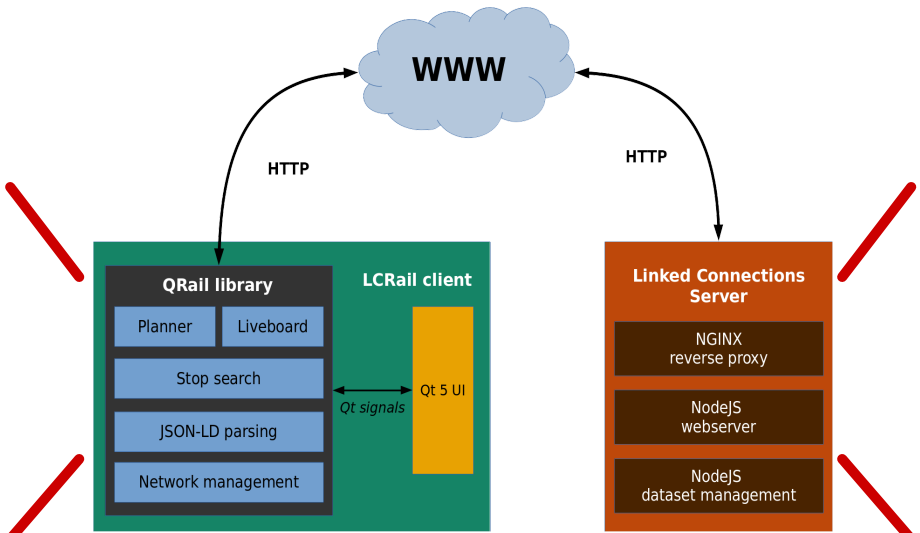
Experiments

Conclusion

Architecture

Self-written LC library & client.

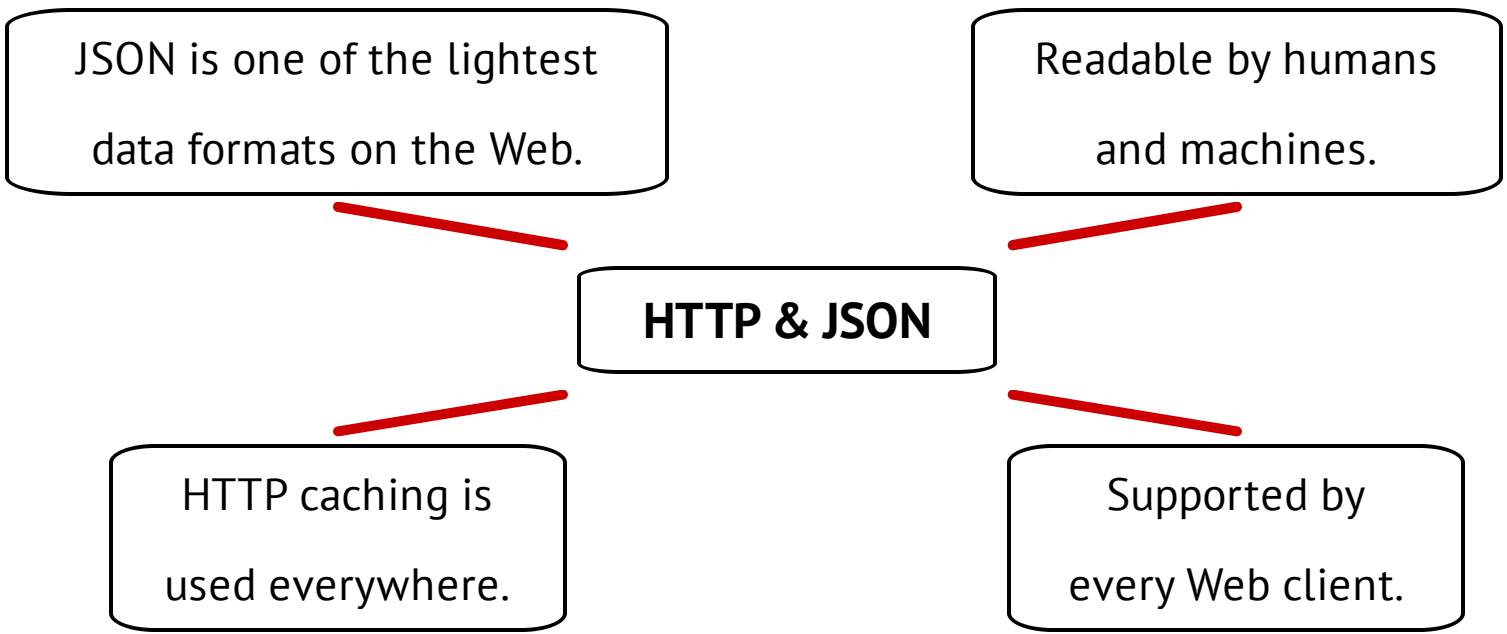
Converts GTFS(-RT) to LC (`gtfsrt2lc`).



Local processing of journeys and liveboards.

Publishes LC fragments to clients.

Real time Open Data



Adding a real time resource to Linked Connections

LC Server

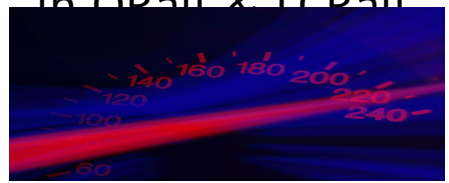
Publishing of the real time data.

QRail & LCRail

Consuming of the real time data

CSA & Liveboard

Efficiently recalculation in QRail & LCRail



Specifications

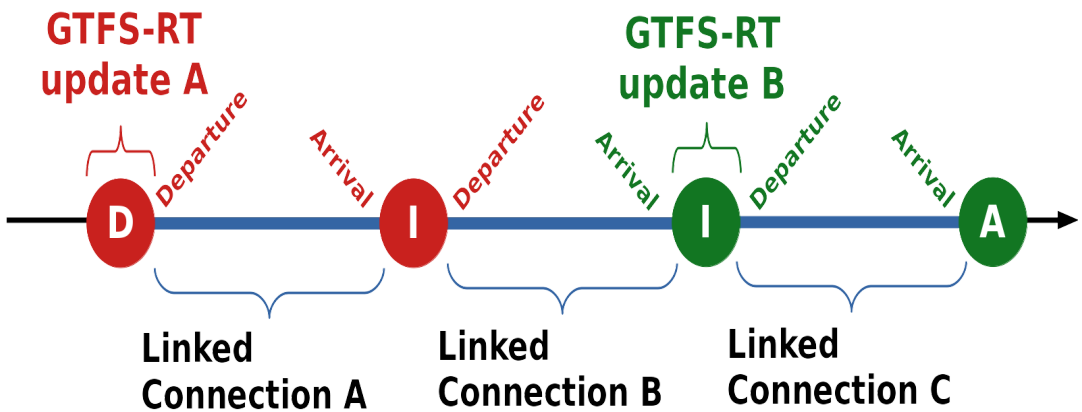
Implementation: LC Server

Experiments

Conclusion

The `gtfsrt2lc` library

- GTFS-RT defines their data differently from LC.
- Only changed delays are supplied.
- Rewrote the algorithm and added cancellations support.



Events Manager: task

- Creates a LC real time feed.
- All events from the last X minutes.

- Only publishes new events.
- SOSA ontology.

- Clients can query the RT resource.
- HTTP polling or SSE.

Specifications

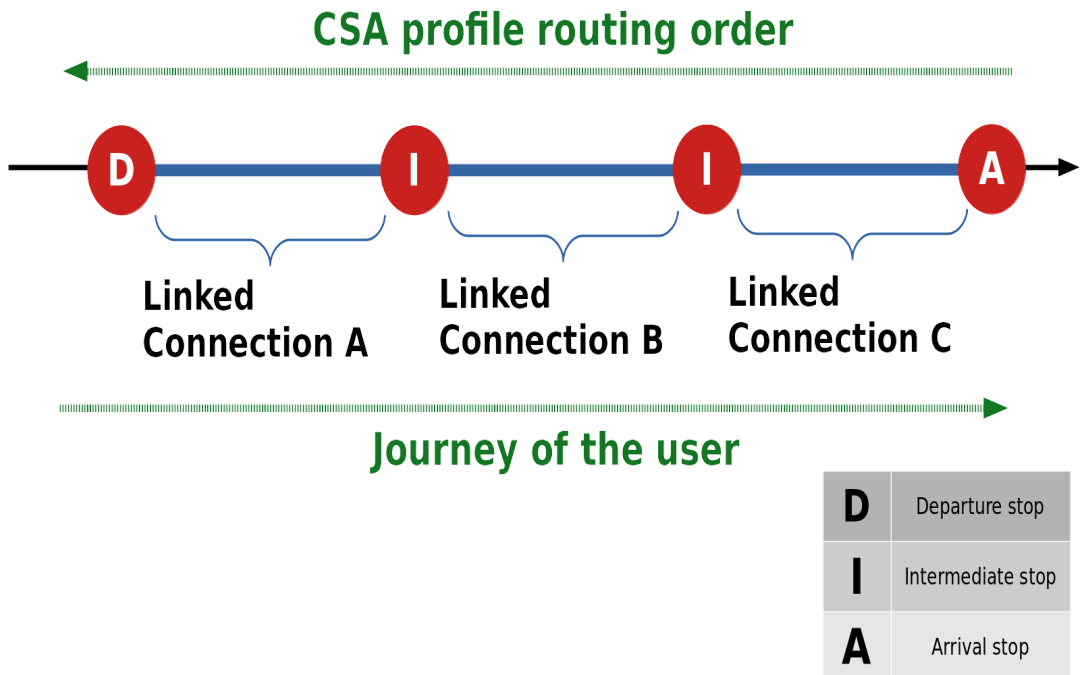
Implementation: QRail & LCRail

Experiments

Conclusion

High level overview of the CSA

- **Scans each connection**: Finds a route from A to B.
- **Profile variant**: Modular with profiles.
- **Faster**: No priority queue (Dijkstra), connections sorted by departure time.



CSA updates in QRail

Combined approach	Snapshots	Rolling back
Filters unreachable connections with CSA EA. Routing is done with CSA profile.	Creates snapshots of the CSA data structures during routing.	Only reroute the changed connections by rolling back to the right snapshot.

High level overview of the Liveboard algorithm

- **Scans each page**: In a certain time frame.
- **URI matching**: Creates a list of all connections matching the stop URI.
- **Previous & next**: The user can get the previous and next results. The necessary pages are automatically fetched.

Specifications

Implementation

Experiments

Conclusion

Test environments

- **Pushing vs polling experiment**: Digital Ocean server & Virtual Wall clients.
- **Processing of updates experiment**: Digital Ocean server & Sailfish OS clients.
- **Conditions**: Each experiment ran for 30 minutes with a clean cache.

DigitalOcean

SAILFISH OS

Specifications

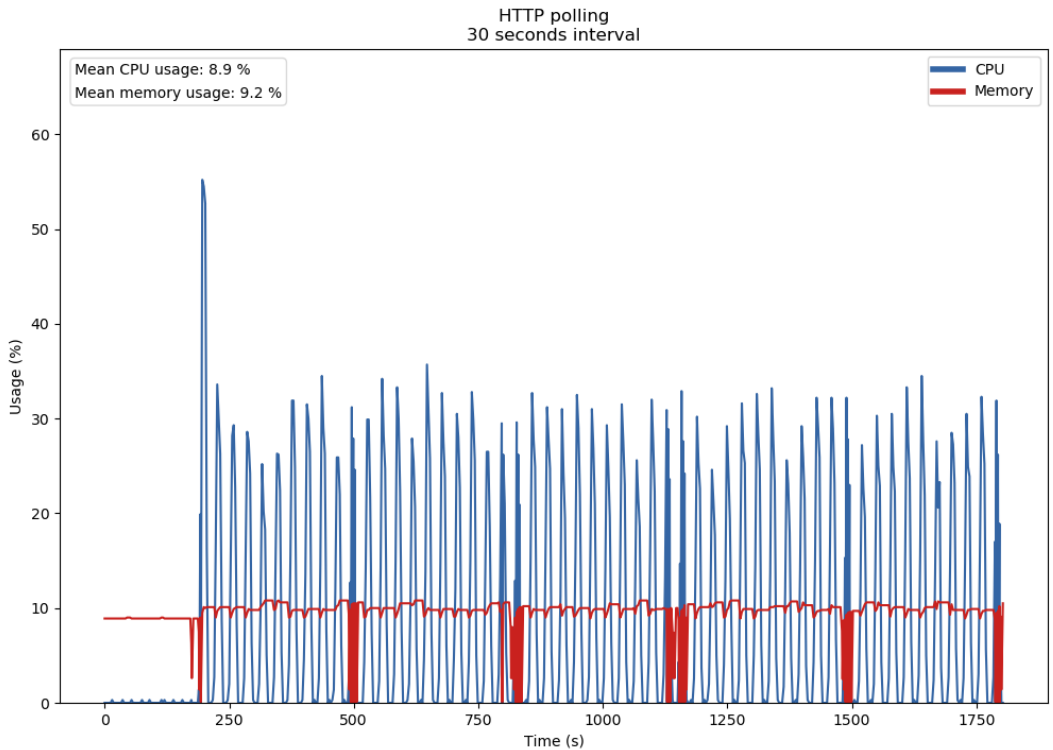
Implementation

Experiments: Pushing vs polling

Conclusion

HTTP polling

- CPU spikes: 35 % when polling.
- RAM usage stays the same.
- Mean CPU usage: 8,9 %.

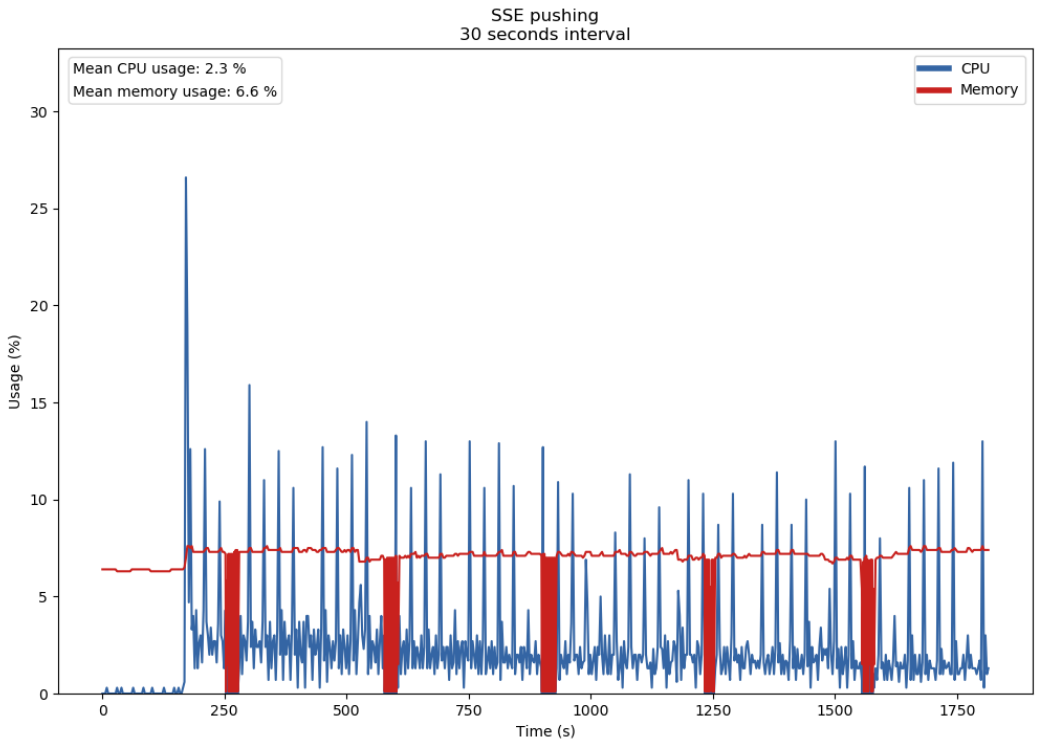


SSE pushing

- CPU spikes: 15 % when pushing.
- RAM usage stays the same.
- Mean CPU usage: 2,3 %.

4x less

mean CPU usage



Specifications

Implementation

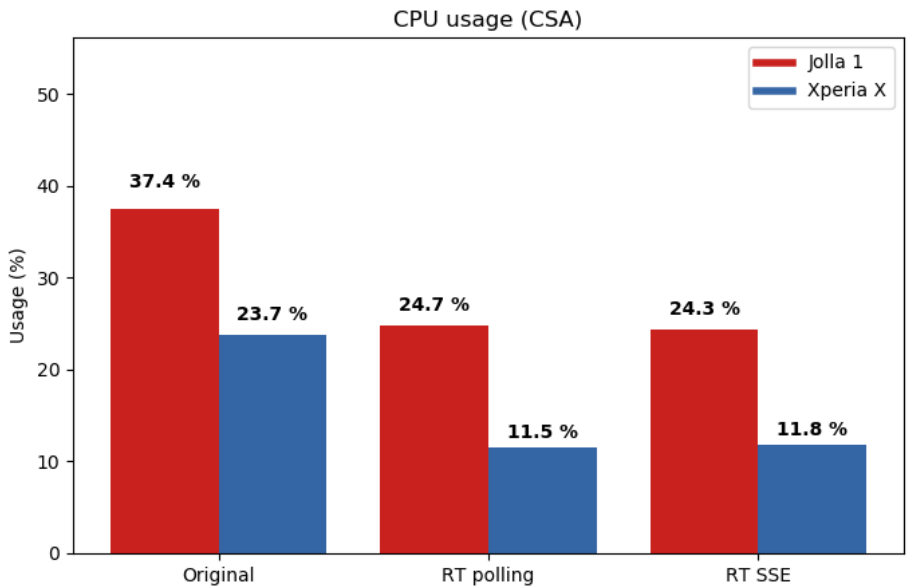
Experiments: Processing of updates (CSA)

Conclusion

CSA: CPU usage

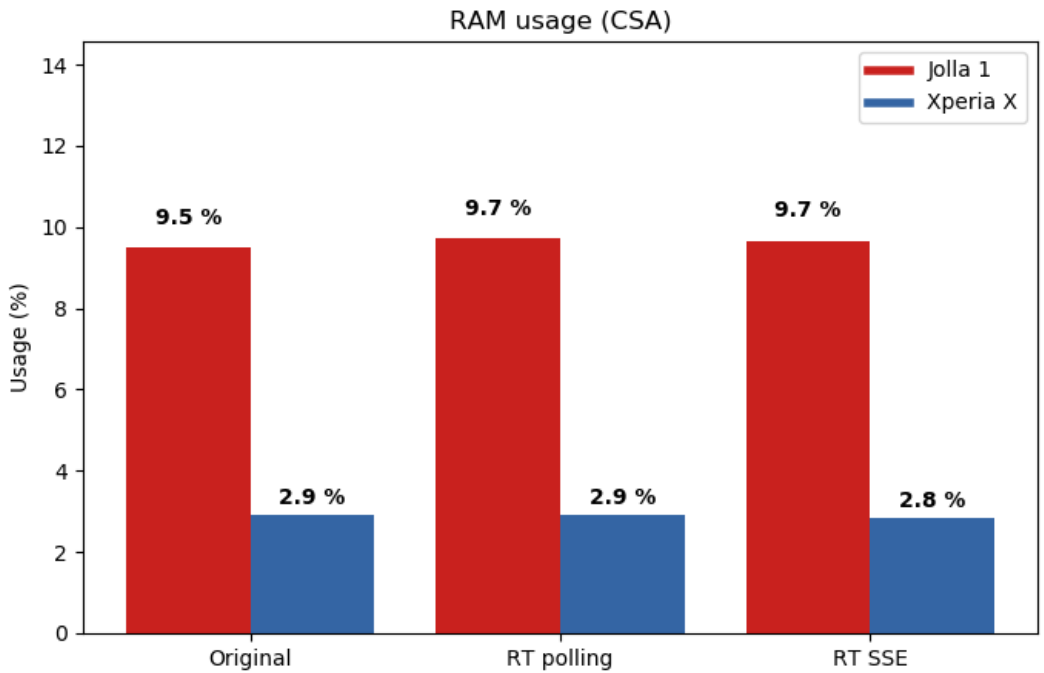
- Hardware plays an important role.
- Snapshot with CSA uses less CPU resources.
- Transport protocol doesn't matter.

Efficiently rerouting



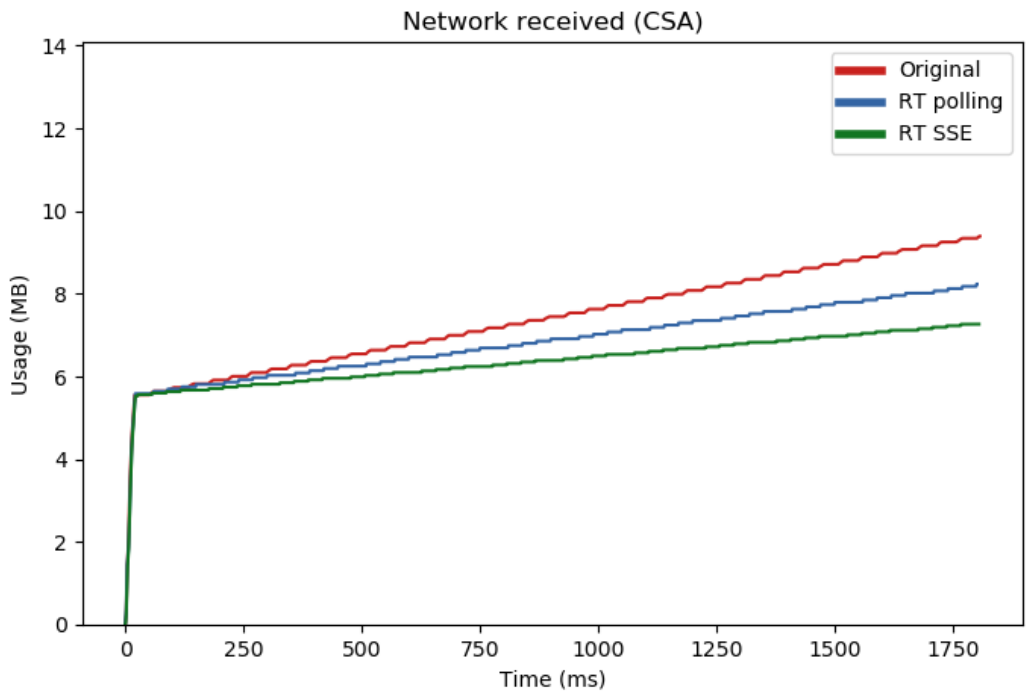
CSA: RAM usage

- Each implementation uses almost the same amount of RAM.
- **Remark**: Xperia X has 3x more RAM than the Jolla 1.



CSA: Bandwidth usage (received)

- **Page cache**: RT resource doesn't need page validation.
- **No events**: RT SSE saves more bandwidth than RT polling when idle.

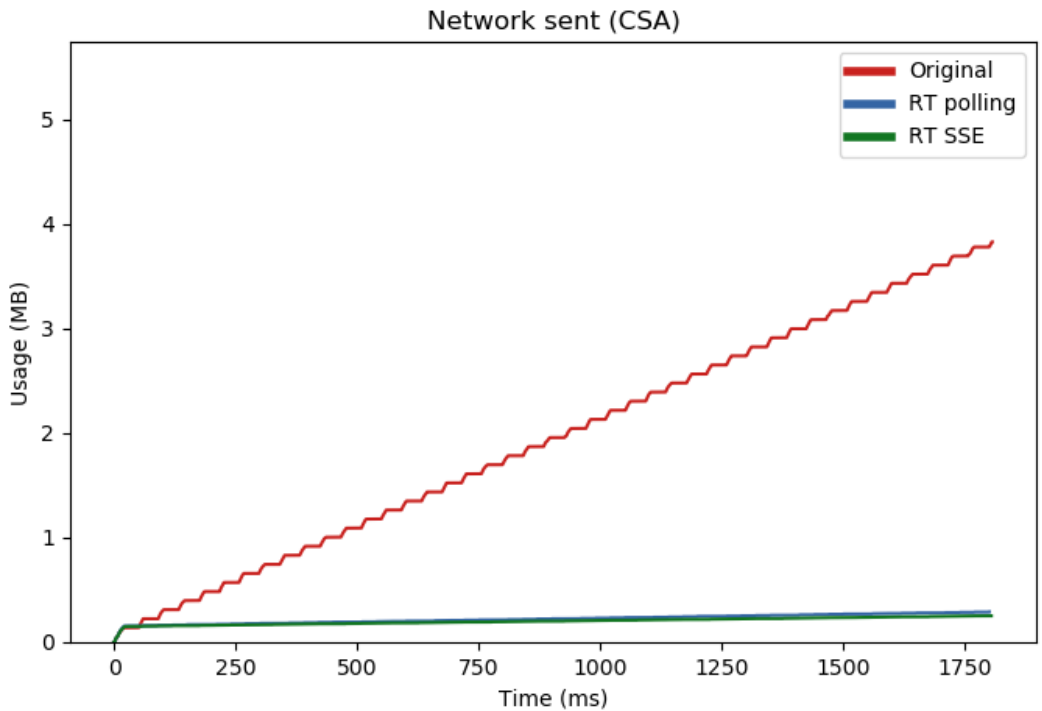


CSA: Bandwidth usage (sent)

- **RT resource**: 1 request for each update.
- **Original**: Validates each page during an update.

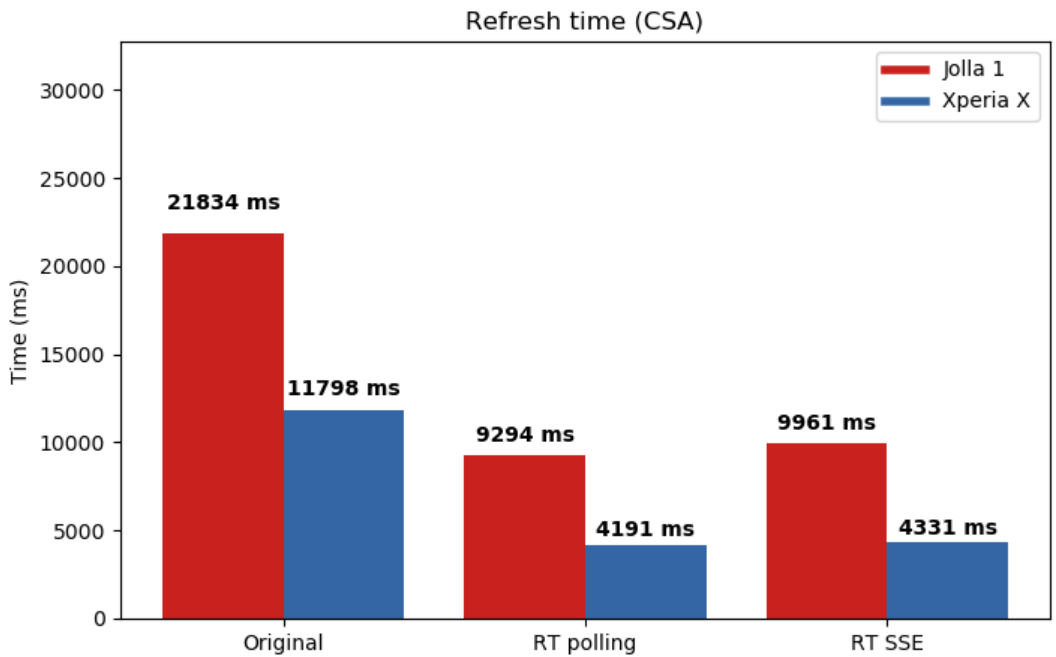
RT resource

saves bandwidth



CSA: User informed time

- Hardware plays an important role.
- Rerouting with snapshots is 2x-3x faster.
- Transport protocol doesn't matter.



Faster informed

Specifications

Implementation

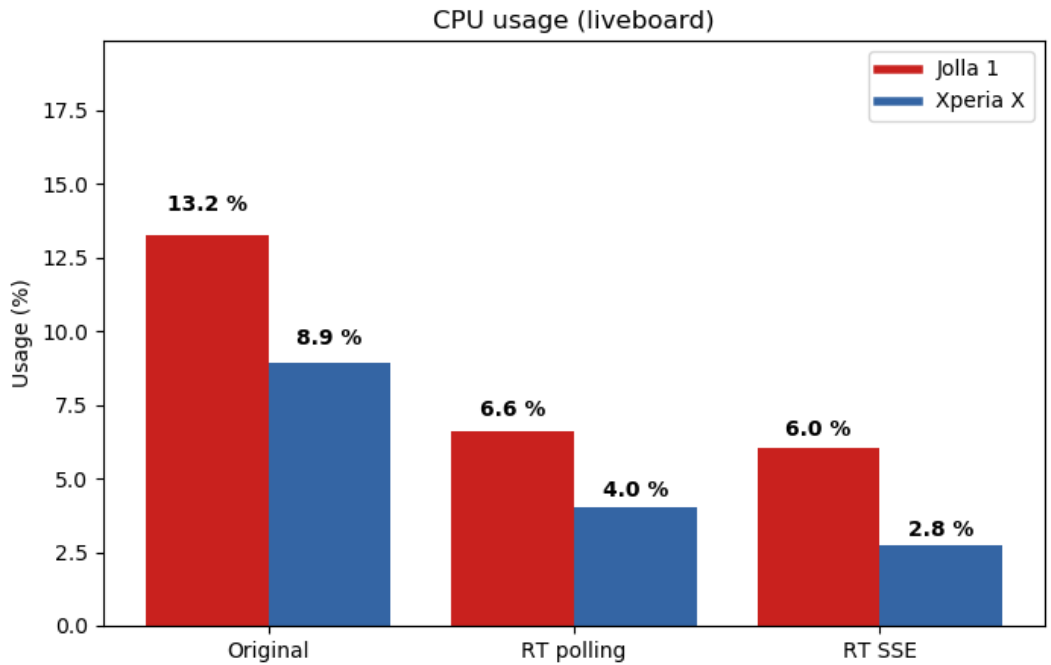
Experiments: Processing of updates (Liveboard)

Conclusion

Liveboard: CPU usage

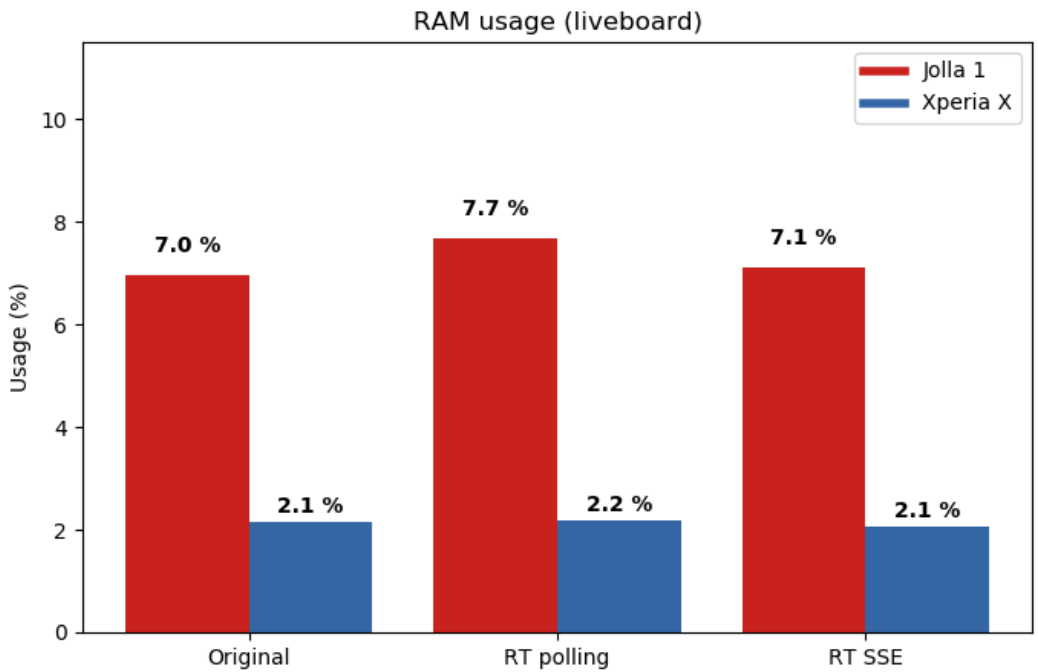
- Hardware plays an important role.
- RT resource uses 2x less CPU resources.
- Transport protocol doesn't matter.

Efficiently recalculating



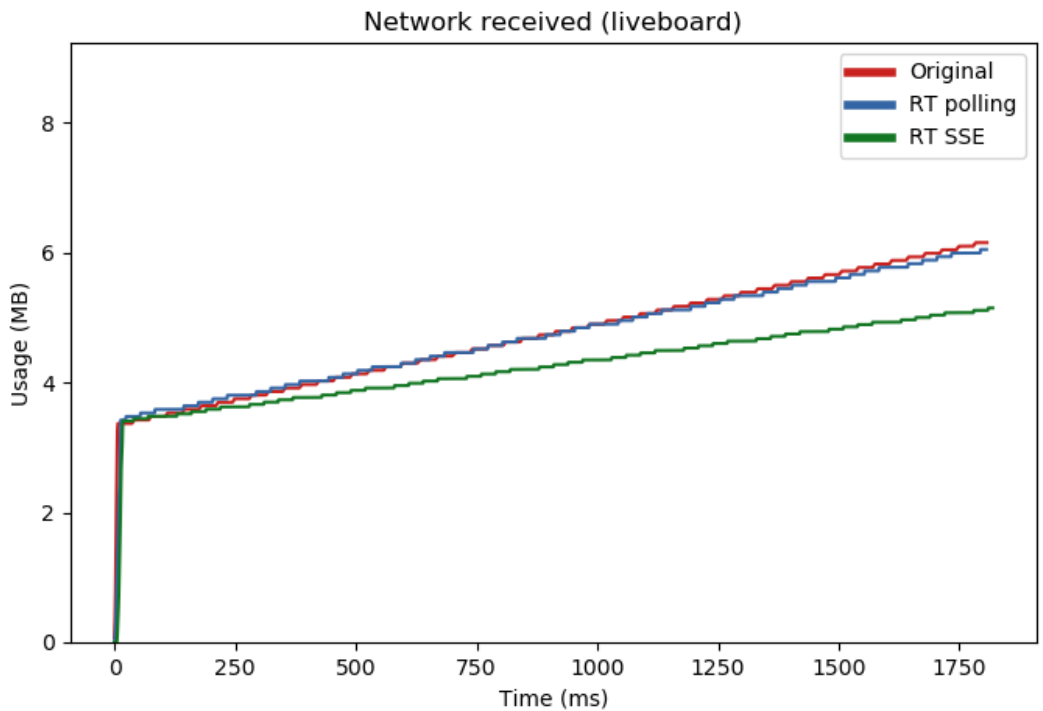
Liveboard: RAM usage

- Each implementation uses almost the same amount of RAM.
- **Remark**: Xperia X has 3x more RAM than the Jolla 1.



Liveboard: Bandwidth usage (received)

- **Page cache**: RT resource doesn't need page validation.
- **No events**: RT SSE saves more bandwidth than RT polling when idle.

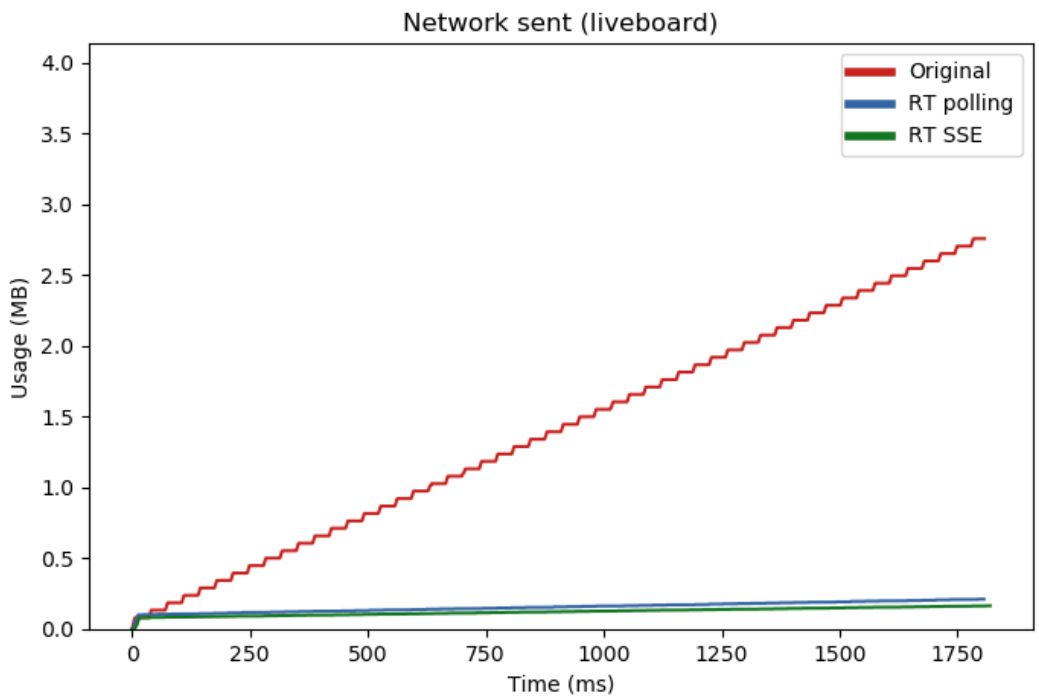


Liveboard: Bandwidth usage (sent)

- **RT resource**: 1 request for each update.
- **Original**: Validates each page during an update.

RT resource

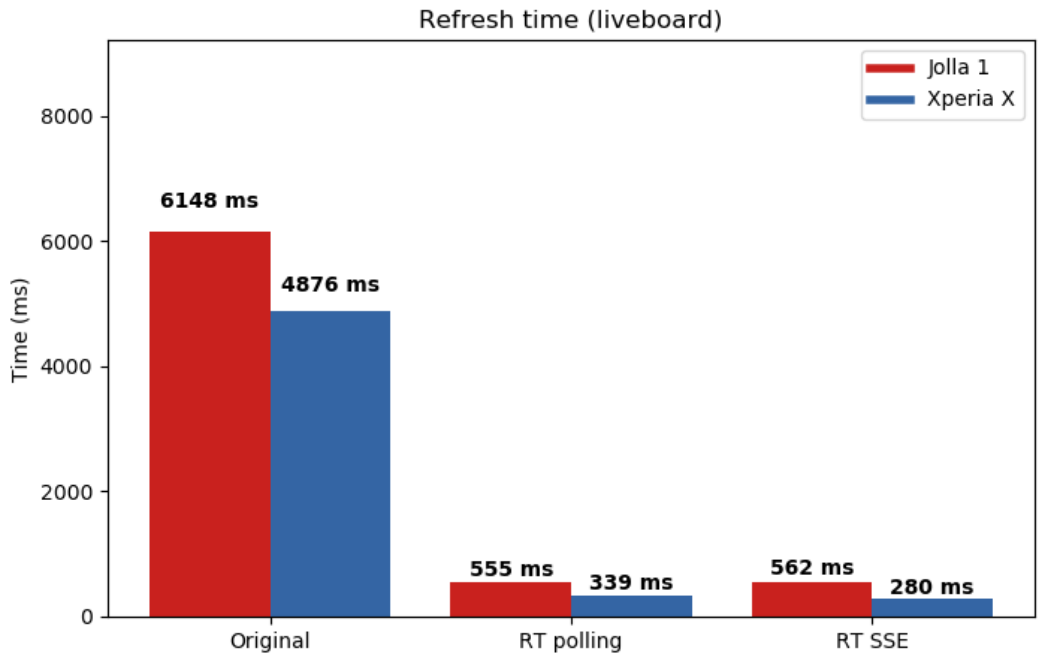
saves bandwidth



Liveboard: User informed time

- Hardware plays an important role.
- Recalculation is 10x faster.
- Transport protocol doesn't matter.

Faster informed



Specifications

Implementation

Experiments

Conclusion

How can we keep a cache for route planners up to date in a cost-efficient way for both Open Data publishers and consumers?

1. **Pushing**: Use a pushing approach like SSE.
2. **Real time resource**: Less bandwidth and processing power is needed.
3. **Updates only**: Only process the changes to the data. Validation is not needed anymore.

How can we consume real time Open Data updates in route planner algorithms in an efficient way?

1. **RT data processing**: Additional logic is needed.
2. **Updates only**: Processing only the changes to the data, improves the CPU and bandwidth usage.
3. **Faster computation**: The user is informed faster about changes to his or her trip.

Achieved goals

1. Notifications keep the user informed in real time.
2. Automatically rerouting using real time data.
3. Less resources consumed for publishers and consumers.

Thank you for your attention!

Are there any questions?

linkedconnections.org

Demo: CSA updates

- **Initial routing**: Caching all pages in the page cache.
- **Update 15h43**: + 4 minutes delay.
- **Update 14h13**: + 1 minute delay.

Demo: Liveboard updates

- **Initial calculation**: All pages are fetched and cached in the page cache.
- **Update 14h33**: + 5 minutes delay.