Crowd-Sourcing Realtime Data

Why?

Current state:

- Free software routing engine (OpenTripPlanner, MOTIS)
- Free software public transport app (Öffi, Transportr, Bimba, KDE Itinerary etc.)
- International free software routing deployment (Transitous)

What’s next?

- Moving the open / proprietary boundary further
- Minimum-viable operation software suite - “community bus stack”?
  - Schedule creation
  - Digital Signage
  - **Realtime data processing**

Context

- Some operators don’t publish delay data, or don’t know it themselves

Plan

- Collect vehicle positions directly from travelers
    - Of course opt-in per trip
    - Regularly send GPS position
- Interpret GPS-position and calculate delay

Collecting Positions

- GPS reception in vehicles
    - With modern phones, multiple satelite networks, AGPS, wireless-network-based geolocation allow roughly correct geolocation in most vehicle types
    - Trains often feature onboard portals that allow to retrieve the position from the train's GPS.
        - Used in microG, KDE Itinerary

Collecting Positions

0. User has enabled crowd-sourcing feature
1. User searches for trip
2. App remembers selected trip. Either as part of the regular functionality, or only for this feature
3. Once start time is reached, app asks for confirmation that user is on the vehicle, and agrees to share location.
4. GPS position and trip id is regularly sent to API

Collecting Positions

- API server exposes positions as a regular GTFS-RT vehicle positions feed
  - Can be consumed by delay calculation
  - No difference from operator-supplied vehicle positions at this point

Interpreting Positions

Prior Art: Transitclock

- Good predictions
- Hard to set up
- Not actively maintained
- Resource intensive
- Relatively unreliable

Interpreting Positions

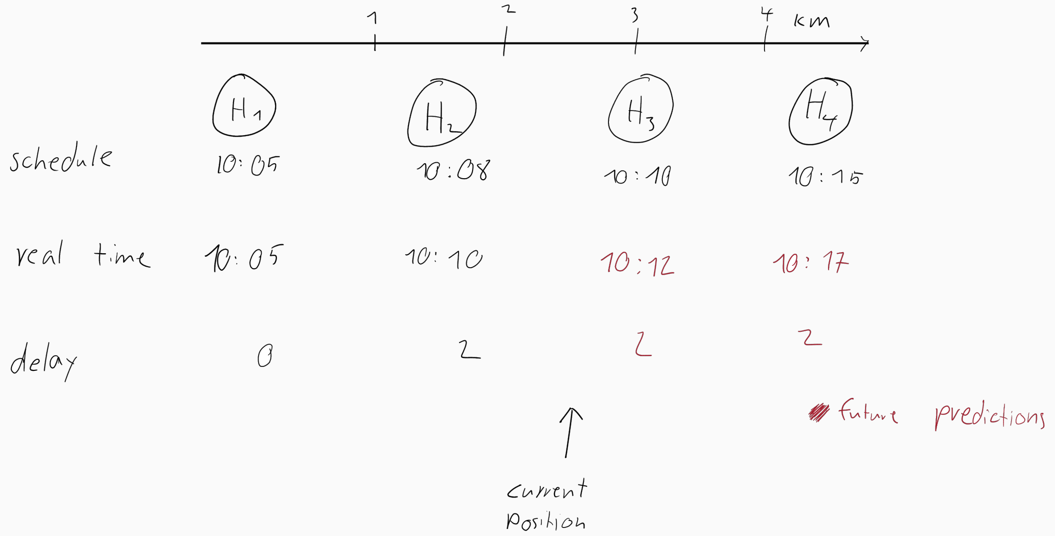
For planet-scale:

- Start with a very simple implementation
- Optimize data model for little resource-comsumption

Interpreting Positions: Simple Implementation

- On new position:
    - store position to buffer
    - if close to stop:
        - delay at stop = position timestamp - scheduled time
        - propagate delay, catching up not modeled
    - if no delay:
        - store buffered positions. That will provide more (time, position) pairs that can be used just like stops
    - else:
        - clear buffer

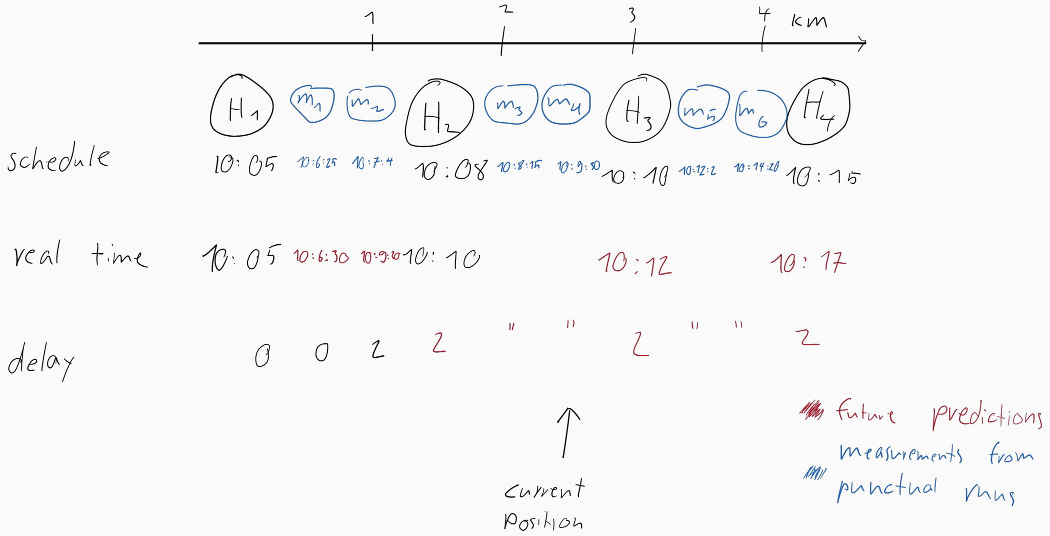
Interpreting Positions



	H_1	H_2	H_3	H_4
schedule	10:05	10:08	10:10	10:15
real time	10:05	10:10	10:12	10:17
delay	0	2	2	2

🟥 future predictions

Interpreting Positions



Interpreting Positions

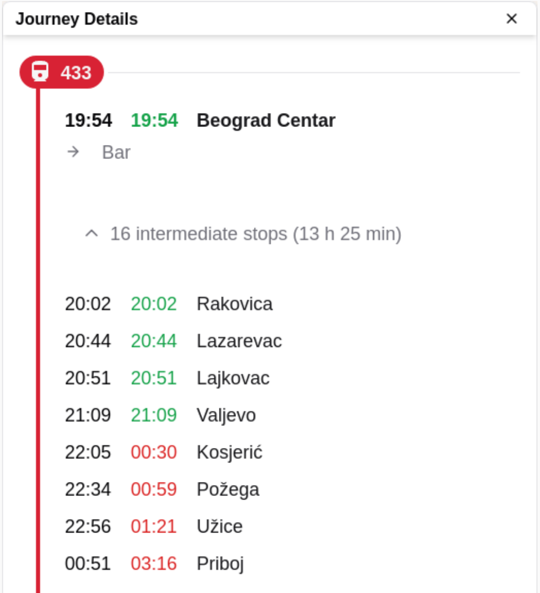
- Expose standard GTFS-RT trip-update feed
    - Can be consumed by standard routing software (MOTIS, OpenTripPlanner)

Not-Live Demo

```
POST http://localhost:3000/api/v1/submit
content-type: application/json

{
    "position": {"lat": 43.58199, "lon": 19.52472},
    "motis_trip_id": "20260127_19:54_me-zpcg_235",
    "timestamp": 1769566648
}
```

Not-Live Demo



Journey Details

433

19:54	**19:54**	**Beograd Centar**
→	Bar	

16 intermediate stops (13 h 25 min)

20:02	20:02	Rakovica
20:44	20:44	Lazarevac
20:51	20:51	Lajkovac
21:09	21:09	Valjevo
22:05	00:30	Kosjerić
22:34	00:59	Požega
22:56	01:21	Užice
00:51	03:16	Priboj

Next Steps

- Test instance deployed in Transitous
  - Use in production
- Client implementation in Bimba?