Backup Tracker Design Document

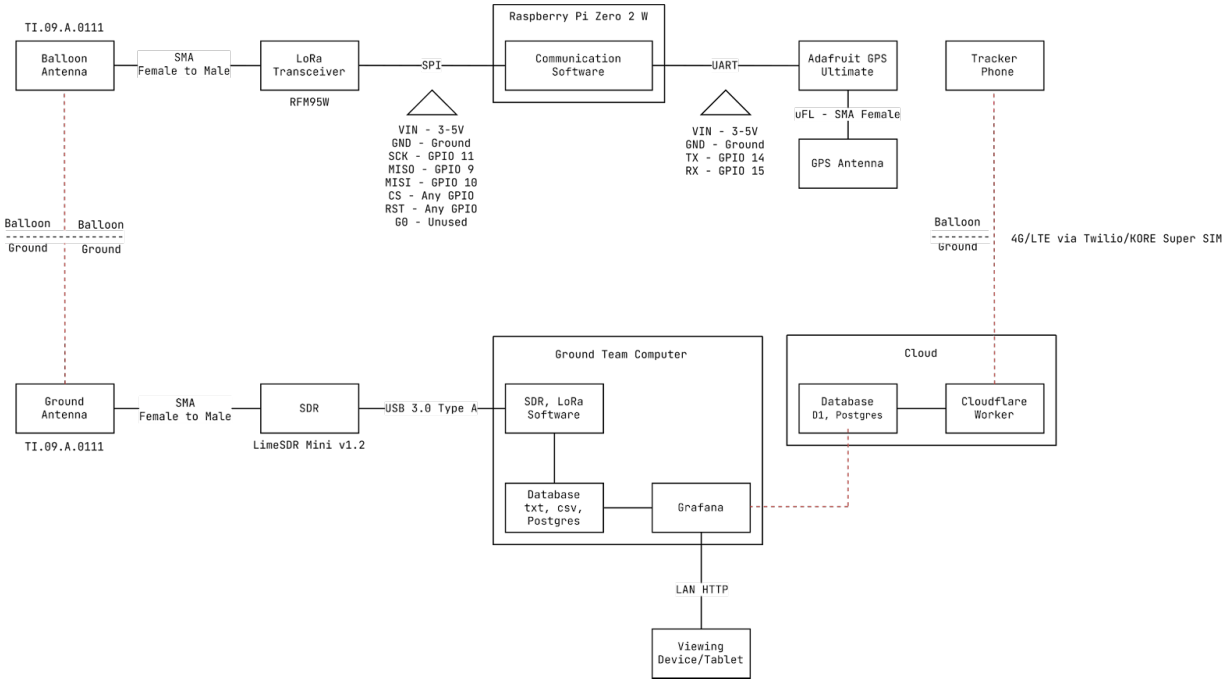
Evan

Preface and Status

This document outlines and discusses important decisions in the design of the backup tracker. This is intended to aid anyone integrating, recreating, or expanding on Balloon 4's backup tracker.

This is a living document and may change at any time, owing to clarification or developments.

Overview



This overview assumes Computing will opt into using the Grafana instance to display information from the main GPS system. At the time of writing, the ground station computing setup is unclear and the depiction above is only one possible route.

Phone

A phone, aboard the balloon, is responsible for collecting and sending telemetry.

Power

Storage

Using the internal battery of the phone is quite desirable as it allows Android to access metrics such as the battery level and temperature. However, owing to the use of an old phone, an

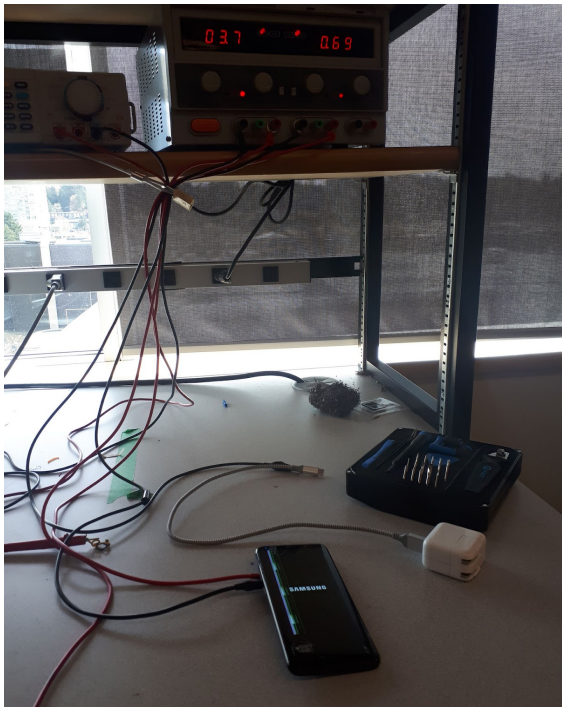
external power bank is a cheap way of replacing an aging internal one. This also alleviates the issues with doze mode.

To extend battery life, we can turn off Wi-Fi, Bluetooth, and NFC. All nonessential apps may be removed.

Testing without a Battery

The Samsung S8 came with a swelling battery which was promptly removed. However, the phone appears to require an internal battery to boot. Salvaging the BMS (Battery Management System) off the bloated battery and installing it allows the phone to sometimes boot. It struggles to boot off of just USB alone, sometimes requiring a trip to recovery mode and a further reboot to boot up.

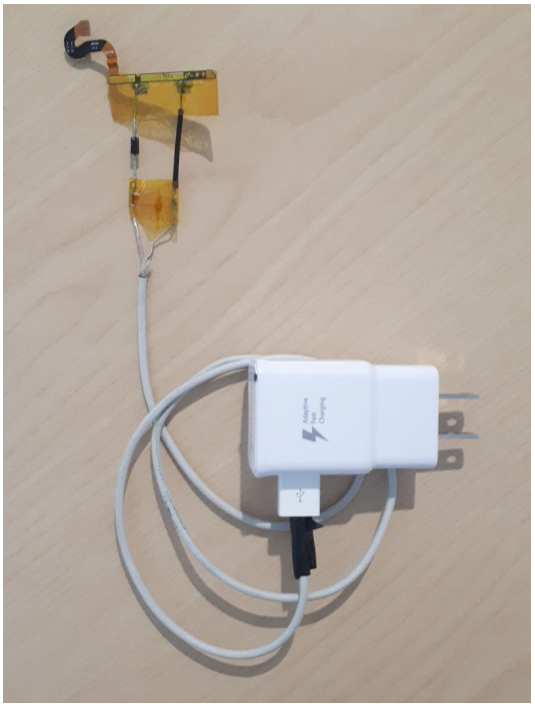
For testing purposes, it is possible to attach a DC bench power supply at ~4V to the leads of the old BMS and boot that way. Note that you need to plug in the device via USB at least one to get it to boot after any disconnection of the BMS or something. This is the forum post that guided this line of inquiry: https://forum.allaboutcircuits.com/threads/samsung-galaxy-s7-power-without-battery.149822/.



Powering the phone via a bench power supply at 3.7V

If you don't have a bench power supply kicking around, there is another solution. Great Scott has an excellent video on powering the phone over USB. I replicated the project using an old USB to lightning cable, a 1N4004 diode, a small wire, some solder, and a bit of Kapton tape.

The diode I used is slightly undersized at 1A continuous - please don't use it. I would go with something like the 1N5408, rated at 3A.



USB to SM-G950W BMS adapter :)

Optimization and Doze

Android's built in power saving features cause issues with the Automate flow if not dealt with. In testing, requests for GPS data would halt indefinitely as power saving features of Android froze the flow/app.

Power savings settings accessible through Android menus didn't resolve this issue by itself.

In testing, advice from https://dontkillmyapp.com/general seems to help. Specifically, disabling idle with **dumpsys deviceidle disable** through ADB worked. On occasion, it appears to still freeze, but that may have been due to a restart clearing the deviceidle status. A lot more testing must be done prior to launch. Note that the command mentioned above must be executed via ADB or with a privileged process on the phone. Automate has some guidance on setting up its privileged process for privileged shell commands.

Temperature

Batteries are quite sensitive to temperature. Especially if the internal battery is used, it should be kept relatively warm. Observe the following:

```
bash -c 'loop() { trap "exit" SIGINT SIGTERM; SECONDS=0; while (( SECONDS < 30 )); do :; done; }; for i in $(seq 1 $(nproc)); do loop & done; wait'
```

This funny little bash script executes a while(true) loop on nproc threads for 30 seconds and then cleans all processes at the end.

By doing so, the CPU is turned into a heater. If the battery is placed onto the CPU, perhaps with copper foil in between, then the CPU can help keep the battery at a reasonable temperature.

Questions:

- Should heating still be enabled if the internal battery is low? What's the threshold?
- What is a good temperature target to turn the heater on at?
- Is this a good idea?

Another solution considered was making an external heater that uses the phone's USB port for power. However, this seems more complicated than necessary and was thus avoided.

To note is that there are plans to have hand warmers in the payload box, possibly alleviating or reducing the need to have heaters on the phone itself.

Moisture

To alleviate moisture issues, conformal coating is probably helpful. See https://youtu.be/qaWvCy2DRSA, where conformal coating is used on a rocket flight computer. This is a design choice that should be made with the help of the Structure team.

Mass

Mass is a significant issue, as there is a prescribed maximum mass that the balloon can be. To reduce mass, the phone can be stripped down to the essential components, removing the frame, cameras, and other unnecessary components.

A power bank, in place of the phone's internal battery, may be unsuitable or undesirable due to the extra mass.

Phone Plan

As a baseline, data capabilities are a must for the tracker to function. While SMS capabilities are nice for error reporting or the like, it isn't completely necessary.

Specialized data-only plans aimed at IoT solutions meet these requirements, with relatively cheap prices at the amount of data the tracker uses.

Physical SIM

Between Twilio/KORE's Super SIM and Hologram's offerings, the former appears to be the cheaper option, with Hologram's shipping cost for a "free" SIM card being $22. It seems that Twilio/KORE also have a free SIM card sort of deal, although the state/region selection is stuck to regions of the United States. Selecting n/a for that field allows you to continue.

Twilio/KORE also only use USD, which seems unideal for any grant reimbursement processes. On the contrary, a strong advantage is that the Super SIM can use the networks of any of the big three (Rogers, Telus, Bell) and also Videotron. This seems quite useful in places with occasionally sparse coverage, such as Carbon, Alberta.

The Super SIM, obtained after much trouble

E-SIM

There *was* an offer for a gigabyte of data for free via an E-SIM.
https://forums.redflagdeals.com/eskimo-travel-free-esim-1gb-data-2-years-2640683/

However, E-SIM requires relatively new devices, making the Backup Tracker more expensive. Unless you can find an E-SIM capable phone for free, this may not be a great option.

If the Eskimo offer had still been valid, the Super SIM still had an advantage. While the Eskimo E-SIM uses the Rogers network, the Super SIM can use the networks of any of the big three (Rogers, Telus, Bell) and also Videotron.

From the CDR, Nick stated that in Carbon, Alberta (planned launch site for Balloon 4), Rogers is pretty bad. Telus and Bell have ok coverage. Therefore, it is best to avoid Rogers.

Implementation

To come.

Usage Restriction

Some cell plans - the Super SIM in particular - are very expensive per MB. There ought to be restrictions in place that prevent unexpected usage.

The Samsung S8 we have can't be rooted or given a custom ROM, and there are no built-in mechanisms to simply disable internet access for a given app.

Amongst the possible options (a decent list can be found by clicking here), a firewall app seems to be the best. No Root Firewall has a note saying to possibly expect no functionality on LTE as IPv6 isn't working there. Given the primary use case of the phone involves LTE, this rules out this app. The other option listed is NetGuard, which seems to check all the boxes.

For NetGuard and almost certainly No Root Firewall, doze is known to be an issue that reduces the functionality of the firewalls. Fortunately, doze optimization is already handled and shouldn't be an issue.

No Enabled Rules

During testing, it seems that a restart also changes the number of allowed rules to 0. It was a simple fix, just enable Automate (and root) in the app. This might be something to watch out for though.

Just tested it again - rules persisted after restart. Something else is resetting the rules.

Telemetry

Payload

The telemetry payloads include longitude, latitude, altitude, date, and other metrics. The exact schema of the telemetry payload will not be included here.

For getting GPS data, there are various providers available to select from in Automate. Among them are:

- Google Play Services location API
- (pure) GPS: This provider determines location using satellites
- High accuracy: The most accurate location available

With Google Play Services location API, see Google Play Services location API vs GPS for details on its shortcomings. TL;DR, it isn't great.

With the remaining two, testing has shown that the high accuracy location provider can be off by a significant amount. Whether pure GPS has the same flaw is unknown. It is hard to test this as

1) It is a little subjective
2) The differences are quite minor and disappear over averages
3) High accuracy will use the pure GPS data if it is fresh, removing any difference

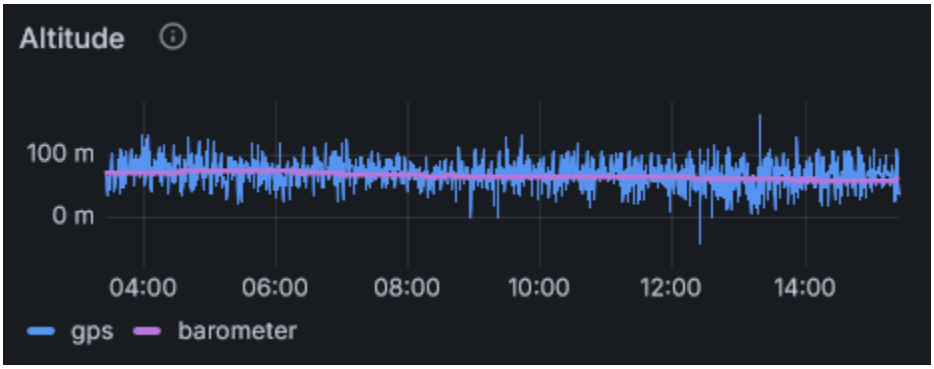
Perhaps it is a bit silly to spend considerable amounts of time on this little detail. If there is time, then this is something one could test. Otherwise, just use pure GPS.

Altitude

There are two main methods to gather altitude information.

The GPS data includes an altitude estimate, but can vary wildly upon successive requests.

The phone has a barometer, which can be used to estimate altitude. In limited testing, this has significantly less fluctuation than the GPS data. However, this estimated altitude is affected by the current weather as air pressure changed with weather. This may be compensable with a change in the formula used to estimate altitude. The current formula used is found at https://rechneronline.de/physics/air-pressure-altitude.php. Using https://rechneronline.de/air/altitude.php, which includes temperature into the formula, ought to be more accurate.



The barometer has much less variance

Collecting and Sending

A “flow” made in Automate is responsible for collecting and sending telemetry. Error handling and resiliency is a guiding principle, ensuring reliability.

The flow is designed such that there are no parallel requests to get and send telemetry. In the event the telemetry runtime freezes indefinitely, that would be catastrophic as no further data could be sent and there would be no easy way of fixing that. Therefore, an upper bound is set on how long each attempt of getting and sending telemetry can be. This is currently set to around 2 minutes. A lower bound is in place as well, with it currently being set to 30 seconds.

The main culprit to timeouts and freezes is in GPS data calls. However, just because GPS data is unobtainable doesn't mean telemetry shouldn't be sent. The telemetry payload still gives a "heartbeat", and can still carry other data like barometer readings, temperature, etc. The solution to this is having a second timeout bound on only the GPS data call.

With no changes in setup, GPS data calls sometimes reach the timeout

On top of having a timeout on getting GPS data, a secondary location provider has been implemented. If the GPS method fails, then a less accurate provider is tried (typically one that uses cell networks to estimate location). The rationale for this is not quite as strong as that for the GPS timeout; there isn't likely to be a scenario where GPS will be unreachable.

Once telemetry is collected, a POST request is made with the payload to the receiving server. SMS was considered, but integration with SMS on the receiver seems overly complicated.

Telemetry is also written to the device throughout the whole mission, even if the HTTP POST request cannot be sent. This data should be pretty fun to look at later.

Requirements

A suitable phone ought to meet the following:

- LTE capabilities
- Unlocked
- Android 4 or higher
    - Android 4 doesn't have the same power saving features of modern Android and thereby might actually work better
- GPS ideally shouldn't have altitude limitations
    - https://gis.stackexchange.com/q/19811
    - This is likely to be very hard to determine
- Ideally has a battery
- Ideally isn't held together with glue

Important settings:

- Battery savings stuff (doze) mentioned throughout

- Data limiting settings, make sure it won't be limited
    - By estimates and very limited testing, the script might use ~500kB of data per hour at 10 second telemetry intervals

The Phone

I acquired a Samsung S8 (SM-G950W) for use with this project. My outline of setting up the phone for use can be found at Phone Setup

It has a few quirks.

- The reported cells strength seems very inaccurate. On my phone, I get a cell strength of about -105 dBm and get a percentage of about 50% cell strength. On the S8, I get about -110 dBm but a percentage of usually 100%.
- The screen has weird software issues where light mode apps were extremely dim. Dark mode apps were fine. It wasn't a consistent behavior either.
- The GPS would (unsure if it still does) sometimes have trouble getting a fix. This is a problem. When it does work though, it is incredibly fast at getting a fix.
- I was concerned it might be a locked phone because it came with the Telus app installed, but it doesn't seem to have any issues with my SIM on the Freedom network.

I reflashed Android onto the S8, but the dimming issue remains. Reflashing SM-G950W.

Operating System

It might've been nice to use a custom ROM, but the bootloader on the GM-G950W is locked. Using a custom ROM would have made it easier to bypass Optimization and Doze issues, especially with root.

Specifically with the S8, the Snapdragon variants have locked bootloaders, but the Exynos versions do not.

Receiver

A backend system is responsible for collecting and storing telemetry sent by the tracker phone.

Database

Postgres was selected as the database of choice. Incredibly popular and reliable, Postgres adapters/projects are available for many other items in the tech stack, including Cloudflare Workers and Grafana.

Cloudflare's beta D1 database was also considered, but owing to limited support in other products like Grafana, Postgres prevailed. Should D1 become more ecosystem rich, its simplicity and integration into Cloudflare Workers is highly desirable.

As of writing, Postgres is exposed to the internet through a Cloudflare Tunnel, protected by Cloudflare Access at https://balloon-4-database.amogsus.com/ with service token authentication. In the production deployment, hosting the database with a commercial cloud provider may be a good idea.

Placing the database in the Cloud™ means a bad internet connection between the ground station and the Edge Computing (likely) won't result in lost data.

Edge Computing

A Cloudflare Worker is responsible for receiving and storing telemetry. You can find the source code at https://github.com/balloon-4/backup-tracker-backend/tree/main.

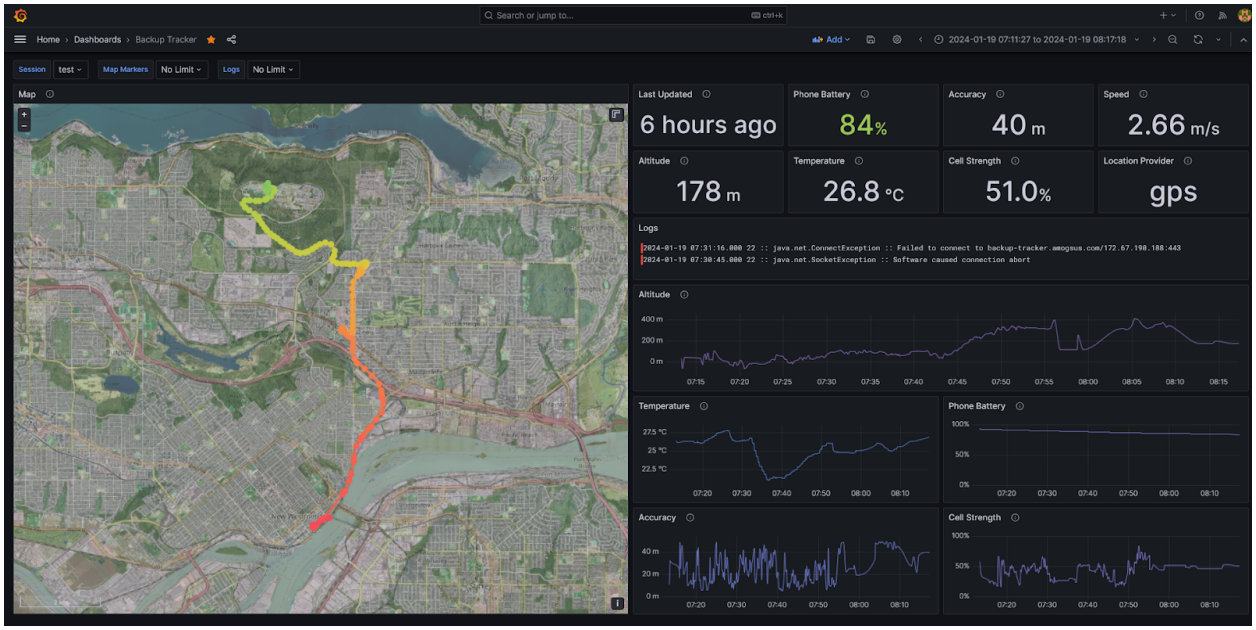
Midware is in place to validate any POST'd payloads. The relevant domain, https://backup-tracker.amogsus.com, is also protected with Cloudflare Access.

The /telemetry and /log endpoint receives telemetry and log payloads respectively. Once received, a Cloudflare Tunnel-compatible Postgres client is used to insert the data into the database.

Frontend

To display the telemetry collected, a Grafana dashboard is used. For Balloon 4, the dashboard is available at https://balloon-4.amogsus.com. If Computing sees it fit, this dashboard can be reused for the main tracking system. Support for the location of the ground team may also be useful.

For Grafana to get the telemetry data, the Postgres data connector is used.



Placement

Grafana is available as both a Cloud™ offering and as a self-hosted application.

Due to the likely limited internet connection, having Grafana with the ground team means the web UI won't have to be sent over the internet. This means it should be more reliable.

However, given the Backup Tracker is a backup system, it isn't particularly necessary to integrate it too deeply into the ground station setup. For simplicity and because I am lazy, it would be preferable to just keep the Backup Tracker as is; all on infrastructure with a good internet connection, separate from the ground station.

Containerization of applications through Docker helps standardize and simplify deployments. As of writing, Grafana is deployed via Docker Compose, allowing for a fairly simple setup. Once started, the dashboard(s) can be imported through the JSON model(s) and is ready to go once the data connector(s) are added.

Geomaps

The Grafana dashboard uses multiple map providers to comprise the map display.

Normally, the client side downloads (parts of) the map and the server has no part in this. However, if the internet connection is severely limited, then this may be an issue.

A solution to this issue is hosting the map server locally, using the xyz tiles option in Grafana.

Stack

XYZ Tile Server

tileserver-gl

Open Street Map

GitHub Gist: https://gist.github.com/attituding/4b1855d47d8b3a2690be11e5d8d52f9d

Summary

Find processed data providers. I used http://download.openstreetmap.fr. After downloading the preferred extract, you get a .osm.pbf file but tileserver-gl requires .mbtiles.

I used Planetiler to convert .osm.pbf to .mbtiles.

```
docker run -e JAVA_TOOL_OPTIONS="-Xmx5g" -v "$(PWD):/data"
ghcr.io/onthegomap/planetiler:latest --osm-path=/data/canada-latest.osm.pbf
--output=/data/canada-latest.mbtiles --download
```

Satellite Map

GitHub Gist: https://gist.github.com/attituding/042e465de022b39a26676a48bb9390b9#mosaic

The Copernicus Programme gives free access to satellite imagery with a resolution up to 10 meters per pixel. Downloading and processing this data via https://browser.dataspace.copernicus.eu is kind of painful and complicated for our purposes. https://s2gm.land.copernicus.eu/mosaic-hub is much nicer, providing a simple way to define regions to download.

Summary

1. Sentinel-2 Global Mosaic (S2GM) provides data as a bunch of small .tif files
2. Use gdal to combine those small .tif files into one big .tlf file
3. Use gdal to convert the big .tif into a .mbtiles file
4. Use gdal to increase the resolution as to increase the maximum zoom level
5. Use gdal to add lower zoom levels

Display

The plan is to use a tablet (or similar device) to display the Grafana dashboard. According to Balloon 3, lugging around a laptop for their setup was unideal. A tablet, especially an ARM based one, should be more than sufficient in terms of performance and battery life.

Possible Changes

- Switch from Automate to a fully custom Android app. This would allow for significantly higher control, including use of accelerometers or other sensors to improve location accuracy.
- A “recovery mode” that emits loud sounds, which could help the ground team find the balloon
- Switch to an edge DB, namely Cloudflare D1. See here.
- Dockerizing Grafana and/or the database for the main GPS system
- Fully automated and complete testing with CI/CD pipelines that test all changes
- Addition and integration of the ground team’s location onto the Grafana map
- Use of Ansible to automate setup