SOFTWARE-DESIGN-PROPOSAL-01 (Omnibus Project Leads)

31 May 2025

Distr. List

Software Lead
Elec Leads
Controls Leads
Omnibus Default Reviewers & Project Leads
Electrical Project Leads

CAN Sources Timestamp Synchronization Design Proposal

Abstract

1. The goal of this memo is to determine a message design that would allow the effective synchronization of messages between the towerside “umbilical” CAN telemetry data (TOWERSIDE) and mission control side “live telemetry antenna” CAN telemetry data (MISSION CONTROLE SIDE).
2. More specifically, the goal of the proposed feature would be to account for the delays within data processing and ensure that for a given message that is received both TOWERSIDE and MISSION CONTROL SIDE, the same message appears with the same timestamp.

Current State

3. Each instance of the parsley source currently acts as independent sources, spitting messages to the 0MQ message queue with a timestamp determined by the host computer’s clock.
4. Here is the current message format in a globallog, once unpacked with msgpack:

```
['CAN/Parsley', 1746896237.84619, {'board_type_id': 'PROCESSOR',
'board_inst_id': 'GENERIC', 'msg_prio': 'LOW', 'msg_type': 'SENSOR_IMU_Z',
'data': {'time': 40.523, 'imu_id': 'IMU_PROC_ALTIMU10', 'linear_accel':
-0.247314453125, 'angular_velocity': -0.244140625}}]
```

Proposed Solution - Chris

5. Use the following algorithm to determine a `calculated_timestamp: int`

   a. Create a hashmap that maps `board_type_id` and `board_inst_id` to their respective initial timestamps., call it `board_to_initial_time_in_ns: dict[str, int]`
   b. If no initial timestamp exists, obtain time from `time.time_ns()`
   c. The `calculated_timestamp` of any given message is calculated by adding `board_to_initial_time_in_ns[board_type_id+board_inst_id]` to the `time` parameter of the board message (converting to the appropriate unit as required)
   d. Create a `board_to_last_time_param: dict[str, float]` where the `time` parameter of each board's clock is recorded in memory
   e. If an overflow or reset is detected, such that `board_to_last_time_param[board_type_id+board_inst_id]` > current time param, flush the value into `board_to_initial_time_in_ns[board_type_id+board_inst_id]` and then add the current `time` parameter; then record into last time param hashmap
   f. With every message, attach both the `calculated_timestamp`, the entire initial times map, and the board to last time param map to allow for synchronization between Parsley sources.
6. Rationale for the above design choices:
   a. A hashmap guarantees `O(1)` access to a given board's initial timestamp, ensuring better accuracy of the relative time calculations. The req'ts do not specify absolute time accuracy, only relative.
   b. Determine a sensibly unique initial time. Use ns to avoid floating point math.
   c. This way we can assign a unique timestamp thanks to the `time.time_ns()` call, which returns a UNIX style timestamp, initially while also relying on the more reliable onboard clock of each individual board instead of basing ourselves on when the message was received to determine message
   d. This is what allows us to detect overflows and resets. See above for reasoning for using a hashmap data structure.
   e. This allows us to keep counting up despite an overflow or a reset. See above
   f. This allows a second Parsley source to synchronize with an existing one. Since the determination of the current timestamp relies exclusively on arithmetic operations, assuming that initial values are the same, a given event will have the same timestamp anywhere (or almost).
7. The synchronization process will be as follows:
   a. The Parsley shall take a new flag that determines whether it starts as a PRIMARY source (does not attempt initial synchronization) or a SECONDARY source (attempt synchronization)
   b. If started as a SECONDARY source, it will instantiate an additional `omnibus.Receiver` to listen for any message from an existing `CAN/Parsley` source. Receive messages for some time and store them in a map of lists.

   c. Once received, unpack the `board_to_initial_time_in_ns` map as well as the `board_to_last_time_param` obtained from the other Parsley source
   d. Start receiving messages over COM. If a reset was noticed between receiving the synchronization packet and the first COM message (obtain by comparing the received `board_to_last_time_param` map and the current times obtained, if the current is smaller AND its value is not in the array of messages received), re-attempt synchronization. This is to prevent a race condition where we set the initial timestamps, and then while switching over to listening over COM, a board reset or overflow has happened, which can cause the initial timestamps to be completely wrong. Hard cap synchronization at 5 seconds, FAIL instead of providing potentially erroneous synchronization.
   e. If a board doesn't exist in the synchronization packets, just initialize with a current time.
   f. Once all boards have confirmed to be synchronized, proceed as usual.
8. Potential pitfalls & Considered alternatives
   a. The race condition when a reset occurs during a synchronization, addressed through various failsafes and a pre-emptive fail condition.
   b. We could have instead sent absolute timestamps and last timestamps, however that would make us deduce initial timestamps anyways (absolute - time parameter). This would also take more synchronization messages, and does not seem to account for the race condition mentioned above either.
   c. We could compare message content, but that would require a constant connection between the 2 parsley sources, which adds complexity
   d. When we initially populate the map with initial times, there may be up to 16 ms of desync between the different boards due to the imprecision inherent in `time.time()`. This is not a problem as 16ms is minor when we measure events in seconds and absolute time is not important since these 16ms will propagate to secondary sources.
9. Misc. Changes
   a. Add the Parsley instance ID to the CAN/Parsley msgs so that we can still parse them as separate series if necessary

Please drop any further proposals down here.

1. Is it better to use `time.monotonic_ns()` instead of `time.time_ns()` since its guarantee is always increasing?
   a. Chris: No, because then we lose the guarantee of uniqueness and it makes timestamps not comparable between different sources, say NI and Parsley. Lets say we were to restart the source when logging is still happening, it would render all the data garbage. With time.time_ns(), we would lose accuracy when restarting but the data will largely be OK. This still reminds me tho that we should absolutely **PUT A BATTERY IN THE FRAMEWORK TO MAINTAIN ITS RTC**. Even if real time accuracy isn't important, it makes everything **convenient.** With

or without the CMOS battery, it doesn't auto-start regardless. We might as well just add a power button or deal with the consequences of our actions.