Status of the acoustic positioning system

Alexander Avrorin

Institute for Nuclear Research

03 February 2023

Initial version of the new APS

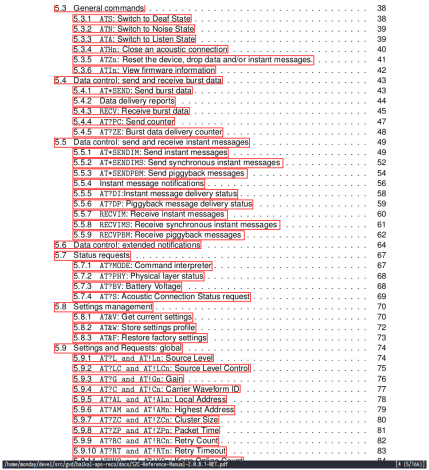
- Near completion (see following slides)
- Written entirely in Python, repo at https://git.jinr.ru/Baikal/baikal-aps-reco
- Diagnostic plots built in R, not required
- Full control over polling, can prioritize more mobile AMs
- Extensive logging
- Distributed: can run reconstruction on separate machines

2 Components: poll control + reconstruction:

- Poll control: connecting to AMs, performing and storing measurements.
- Reconstruction: producing equivalent of `measurements*` files (one per modem)

Poll control: basic principle

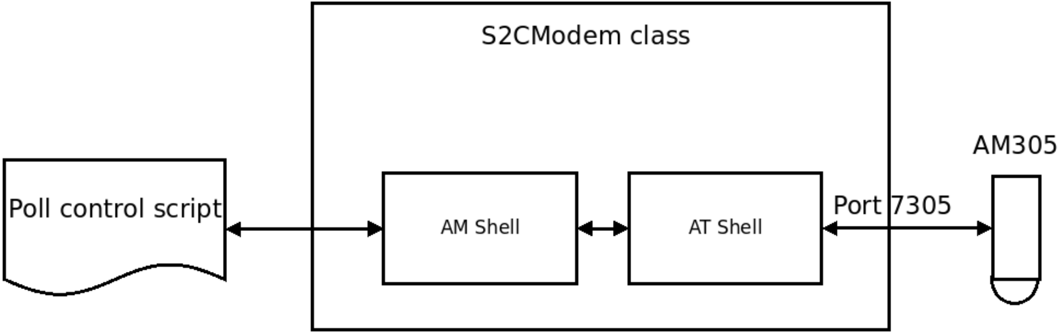
- All EvoLogics AMs have a Linux OS on board (we call Linux command shell AM shell).
- Linux shell for modem N is available on unit 1 at port 7000 + N
- Direct AM control is possible via a separate AT shell that is available locally via port 9200 or 9201.
- List of AT commands is available with AM docs (see ‘doc‘ directory in repo)



/home/monday/devel/src/gvd/baikal-aps-reco/docs/S2C-Reference-Manual-2.0.B.1-NET.pdf [4 (5/166)]

Poll control: architecture

- All communication with AM performed via `S2CModem` class
- `S2CModem` can receive commands corresponding to AT commands and provide structured response
- Communication failures, command failures etc taken care of
- Far from all of the AT commands added, but the essentials are there

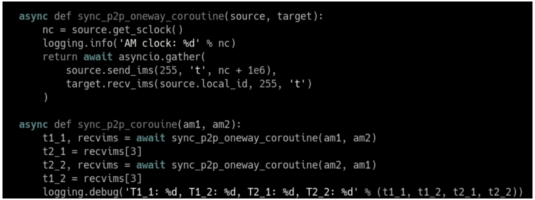


Poll control: using async modules

- Newer versions of Python have async/await functionality
- It allows launching several independent subroutines, then waiting for them to complete
- We use it to control multiple AMs simultaneously

Code to the right:

- Source AM broadcasts message ’t’ over acoustic channel
- Target AM receives this message
- ‘await‘ means we wait for both of this conditions, so we can extract transimission timing info



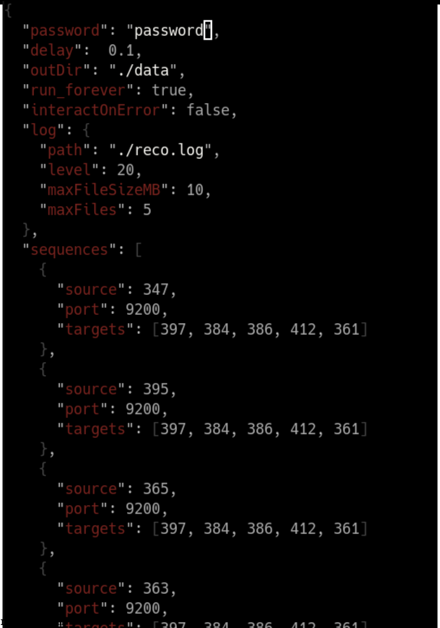
```
async def sync_p2p_oneway_coroutine(source, target):
    nc = source.get_sclock()
    logging.info('AM clock: %d' % nc)
    return await asyncio.gather(
        source.send_ims(255, 't', nc + 1e6),
        target.recv_ims(source.local_id, 255, 't')
    )

async def sync_p2p_corouine(am1, am2):
    t1_1, recvims = await sync_p2p_oneway_coroutine(am1, am2)
    t2_1 = recvims[3]
    t2_2, recvims = await sync_p2p_oneway_coroutine(am2, am1)
    t1_2 = recvims[3]
    logging.debug('T1_1: %d, T1_2: %d, T2_1: %d, T2_2: %d' % (t1_1, t1_2, t2_1, t2_2))
```

Poll types: sequential

We have 2 poll scripts so far: syncpoll and seqpoll

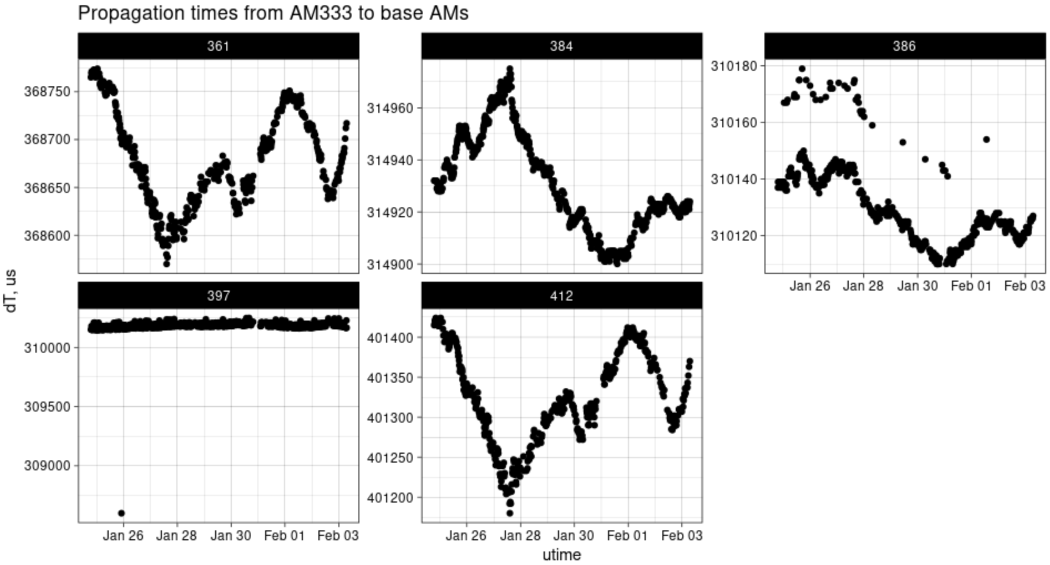
Seqpoll: sequential polling with acknowledgement

1. Connects to all AMs 1-4 (currently clusters 9-11)
2. From each connected AM it measures prop time to selected base nodes
3. Measurement performed with acknowledgement, not affected by clock drift
4. Can work with interference from Kebkal polls
5. Currently in operation

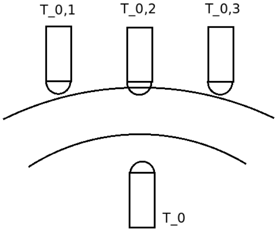


```
{
  "password": "password",
  "delay":  0.1,
  "outDir": "./data",
  "run_forever": true,
  "interactOnError": false,
  "log": {
    "path": "./reco.log",
    "level": 20,
    "maxFileSizeMB": 10,
    "maxFiles": 5
  },
  "sequences": [
    {
      "source": 347,
      "port": 9200,
      "targets": [397, 384, 386, 412, 361]
    },
    {
      "source": 395,
      "port": 9200,
      "targets": [397, 384, 386, 412, 361]
    },
    {
      "source": 365,
      "port": 9200,
      "targets": [397, 384, 386, 412, 361]
    },
    {
      "source": 363,
      "port": 9200,
      "targets": [397, 384, 386, 412, 361]
```

Example data of sequential poll

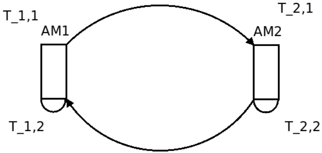


Poll types: async



$$\begin{aligned} T_{2,1} &= T_{1,1} + \delta_{1,2} + \Delta_{1,2} \\ T_{1,2} &= T_{2,2} + \delta_{1,2} - \Delta_{1,2} \end{aligned}$$

$$\begin{aligned} \Delta_{1,2} &= \frac{T_{2,1} + T_{2,2} - T_{1,1} - T_{1,2}}{2} \\ \delta_{1,2} &= T_{2,1} - T_{1,1} - \Delta_{1,2} \end{aligned}$$

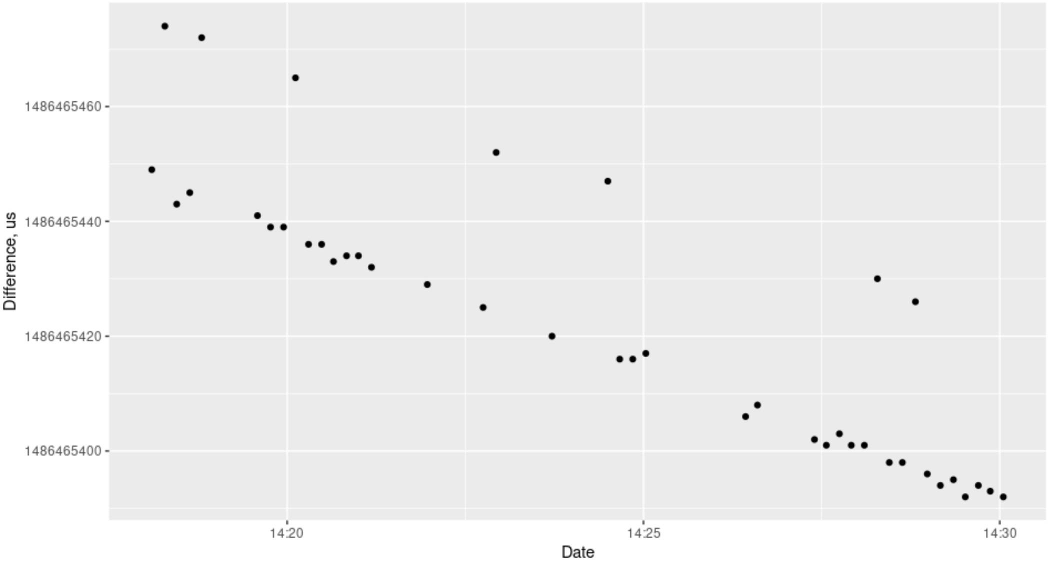


Test results:

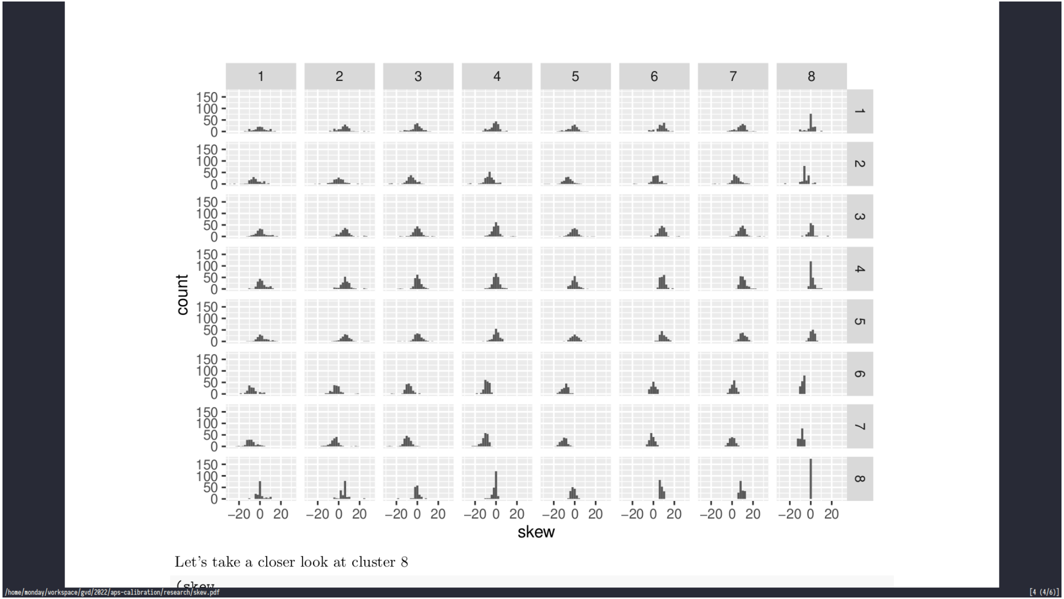
- $\delta_{sync} = 436574, 436576, 436547$
- $\delta_{im} = 436562$

Sync measurements

Slope: about 77 ns/s



Kebkal clock skew



Let’s take a closer look at cluster 8

```
(skew
```

/home/monday/workspace/gvd/2022/aps-calibration/research/skew.pdf

[4 (4/6)]

Comparison syncpoll vs seqpoll

Seqpoll:

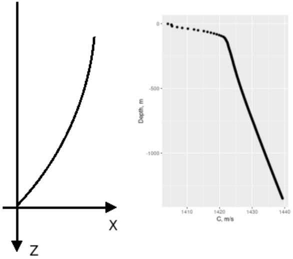
- Complete (could use some work though)
- Works in adversarial environment
- 15 min polling cycle
- Linear growsth with number of AMs expected

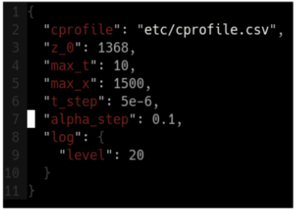
Syncpoll:

- Not yet complete
- Possibly *much* faster polling times
- Poll timing growth with number of base AMs
- Poll timing unvailable. Could be issues with older AMs.

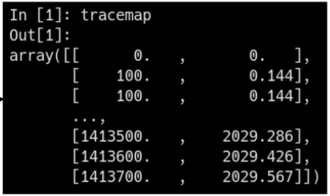
Reconstruction: raytracing

- Trace rays with a fixed angular step
- raytrace.py
- Output in hdf



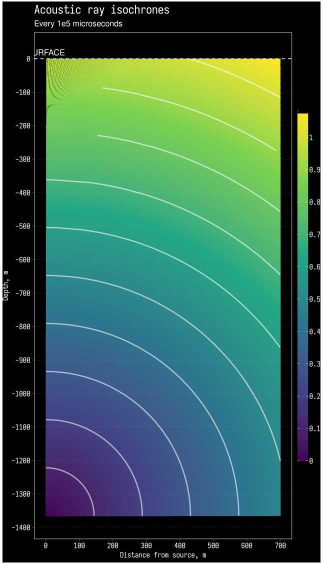


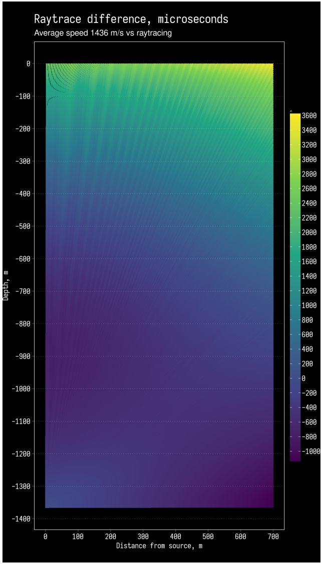
```
{
  "cprofile": "etc/cprofile.csv",
  "z_0": 1368,
  "max_t": 10,
  "max_x": 1500,
  "t_step": 5e-6,
  "alpha_step": 0.1,
  "log": {
    "level": 20
  }
}
```

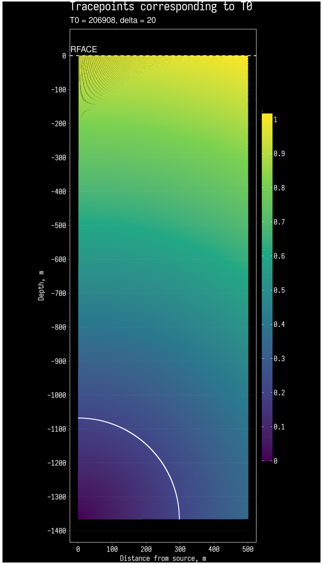


```
In [1]: tracemap
Out[1]:
array([[      0.    ,        0.   ],
       [    100.    ,        0.144],
       [    100.    ,        0.144],
       ...,
       [1413500.    ,     2029.286],
       [1413600.    ,     2029.426],
       [1413700.    ,     2029.567]])
```

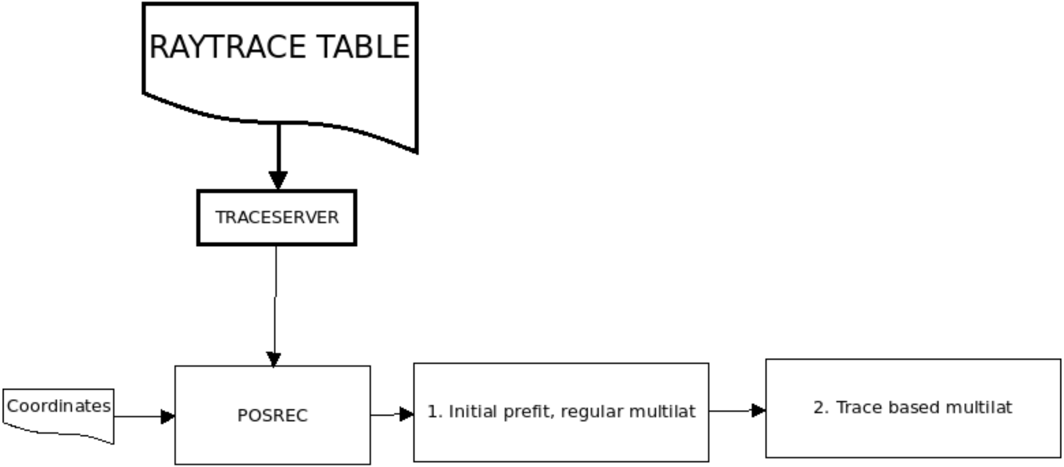
Isochrones


Acoustic ray isochrones
Every 1e5 microseconds
JRFACE
0
-100
-200
-300
-400
-500
-600
-700
-800
-900
-1000
-1100
-1200
-1300
-1400
Depth, m
0
100
200
300
400
500
600
700
Distance from source, m
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Raytrace difference, microseconds
Average speed 1436 m/s vs raytracing
0
-100
-200
-300
-400
-500
-600
-700
-800
-900
-1000
-1100
-1200
-1300
-1400
Depth, m
0
100
200
300
400
500
600
700
Distance from source, m
3600
3400
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
-200
-400
-600
-800
-1000


Tracepoints corresponding to T0
T0 = 206908, delta = 20
RFACE
0
-100
-200
-300
-400
-500
-600
-700
-800
-900
-1000
-1100
-1200
-1300
-1400
Depth, m
0
100
200
300
400
500
Distance from source, m
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Reconstructiom: pipeline


RAYTRACE TABLE

TRACESERVER

Coordinates

POSREC

1. Initial prefit, regular multilat

2. Trace based multilat

To finalize

- We have a solid foundation for a minimally viable APS
- Seqpoll or syncpoll?
- Need to analyze reconstruction
- Need to introduce cuts on signal quality
- Need help