USRP time synchronisation with Octoclocks for distributed nodes, practical implementation and measurements

Cyrille Morin

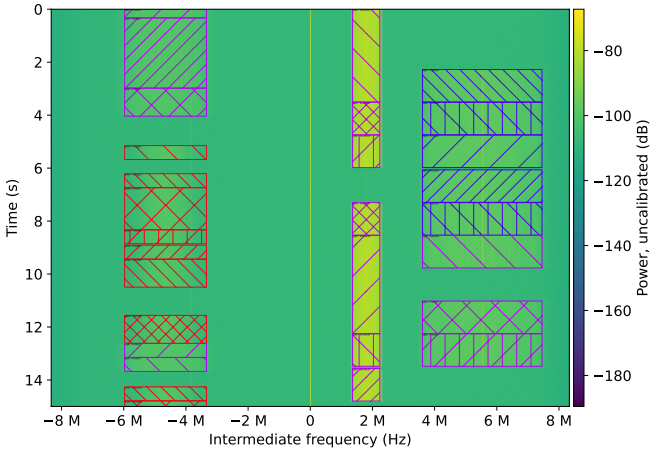
Maracas team, CITI Lab, Inria

29/03/23

Our synchronisation needs

A dataset for wideband signal detection

- Generate scenarios mimicking real spectrum usage
- Annotate time/frequency location of transmitted signals
- Useful as benchmark for signal detection algorithms, and for training ML systems
- Requires common time frame to match transmission time and recieved samples



Objectives

Design goals

- Use the hardware available in the experiment room
- As much as possible, use available APIs

How to measure time error?

- Common question in these talks
- Compare synchronised time frame with timing from preamble detection

Presentation objectives

- Describe implementation details to help with reuse
- Present obtained metrics

Radio nodes in CorteXlab

USRP 2932

- USRP N2xx/B2xx generation
- Corresponds to a N210 with SBX daughterboard
- 400 MHz to 4.4GHz
- Up to 25Msps with 20MHz BW
- 10MHz and PPS input
- Currently 22 installed (+6 in the works)

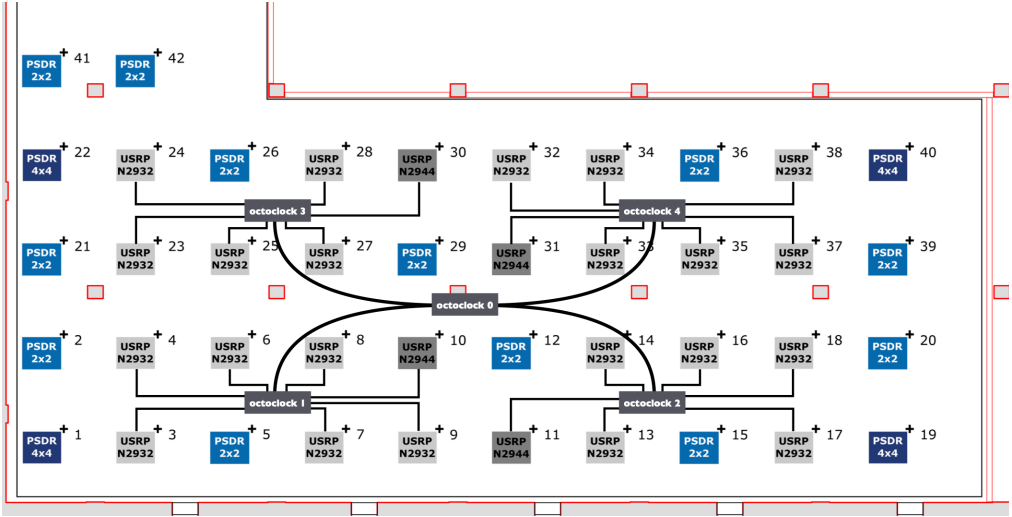
USRP 2944R

- USRP X310 generation
- Corresponds to a X310 with UBX daughterboards
- 10 MHz to 6GHz
- Up to 200Msps with 160MHz BW
- 10MHz and PPS input
- Currently 4 installed (+4 in the works)

Connected computers

- One computer solely connected to one USRP, making one node
- All on a common ethernet LAN, with access to internet
- Requires collaboration between nodes

An Octoclock tree



Tight PPS sync

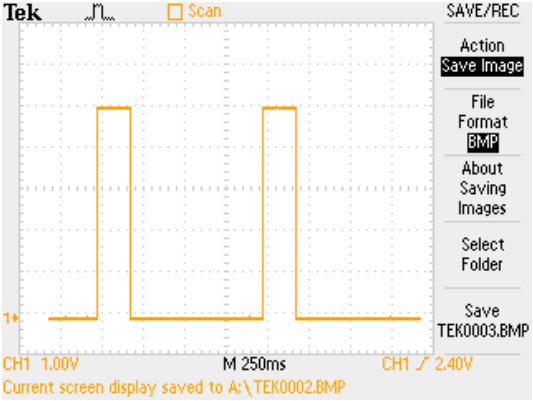


Figure: Oscilloscope capture of two PPS pulses (1s interval)

Tight PPS sync

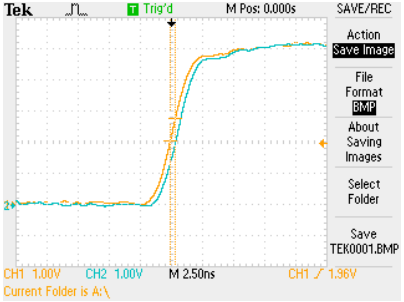


Figure: Time difference between output of Octoclock 4 and 2 (Nodes 31 and 14)

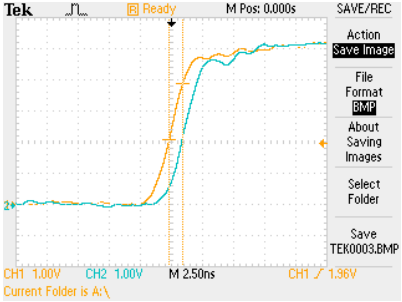


Figure: Time difference between output of Octoclock 1 and 3 (Nodes 10 and 27)

GNU Radio Tags

From UHD Source

- rx_time value is tuple with integer seconds, fractionnal seconds
- Emitted at stream start
- Reemitted after each loss of continuity overflows
- Also provides rx_rate value is sample rate
- Does not trigger on clock reset, would need manual trigger by calling empty stream command
- Manual trigger possible (empty stream cmd), but wrong clock value

To UHD Sink

- Burst mode: define a burst of samples to transmit
- Either provided as a start of burst tag (user defined key) with number of samples in value
- Or as start and end tags (with keys: sob and eob)
- Can add a timing tag at SOB: tx_time, with value a tuple with (integer seconds, fractionnal seconds)

UHD blocks API: time

Get/Set Time

- `uhd_block.get_time_now()` : returns current USRP clock time
- `uhd_block.get_time_last_pps()` : returns USRP clock time when last PPS was received
- `uhd_block.set_time_next_pps(value)`: Next PPS, set USRP clock time to desired value

Time syntax

- Above commands return/expect a time_spec object, not primitive type (time_spec_t in C++)
- Can be created (Python) with: `uhd.time_spec(integer seconds, fractionnal seconds)` (similar to tags)
- Can extract float seconds with `time_spec.get_real_secs()`

Old bug

- USRP 2944 may use PPS falling edge instead of rising edge
- Causes approx 300ms delay between USRP types
- Fixed for UHD with GNU Radio $>$ 3.10

UHD blocks API: Streaming

Rx Stream start

```
def start_uhd_stream(self):
    cmd = uhd.stream_cmd_t(uhd.stream_mode_t.STREAM_MODE_START_CONTINUOUS)
    cmd.stream_now = True
    self.usrp_block.issue_stream_cmd(cmd)
```

Rx Stream stop

```
def stop_uhd_stream(self):
    cmd = uhd.stream_cmd_t(uhd.stream_mode_t.STREAM_MODE_STOP_CONTINUOUS)
    cmd.stream_now = True
    self.usrp_block.issue_stream_cmd(cmd)
```

Snippets

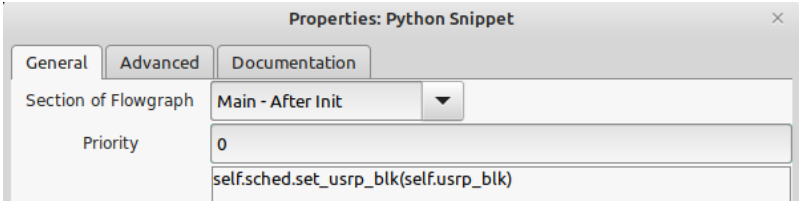
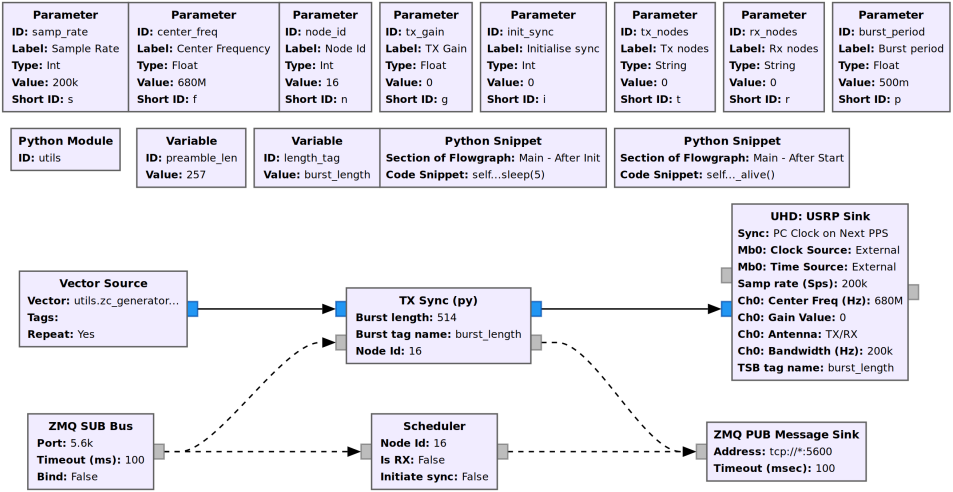
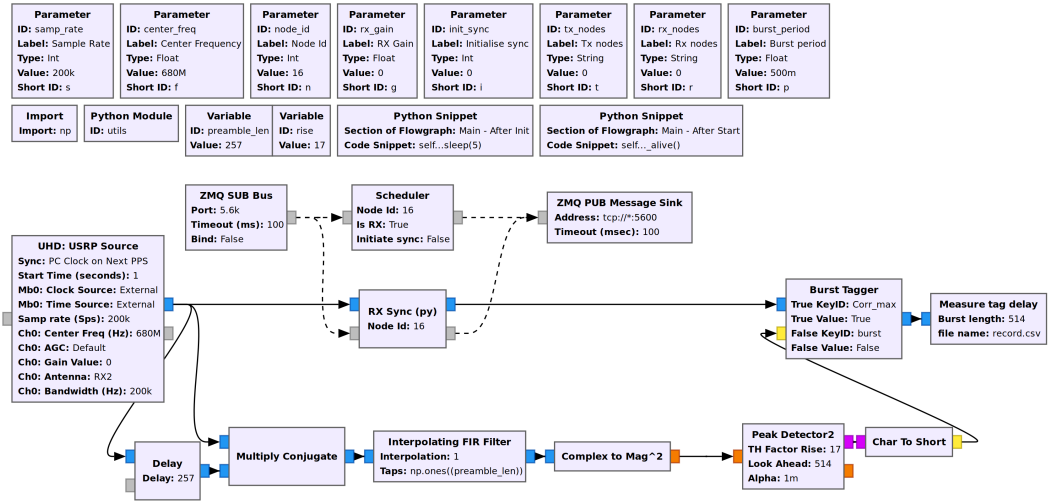


Figure: A way to give blocks access to other blocks
In this case, the UHD Source block has for id usrp_blk

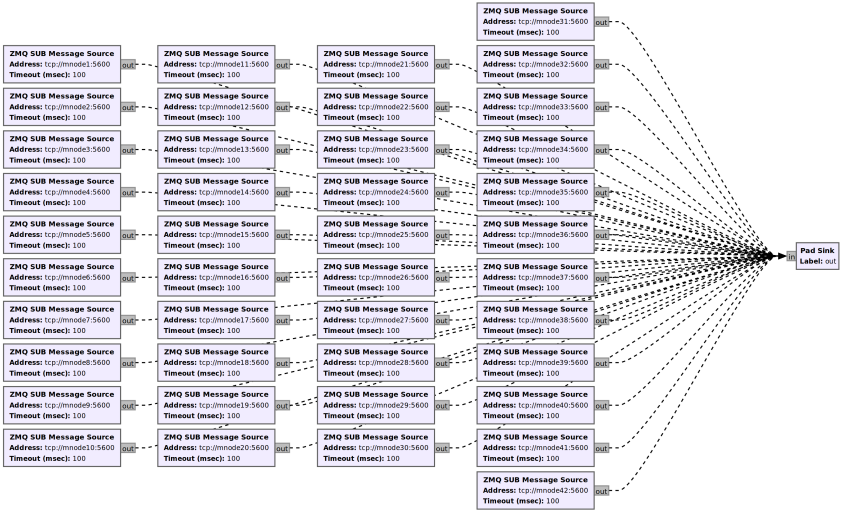
Transmitter



Receiver


Parameter
ID: samp_rate
Label: Sample Rate
Type: Int
Value: 200k
Short ID: s
Parameter
ID: center_freq
Label: Center Frequency
Type: Float
Value: 680M
Short ID: f
Parameter
ID: node_id
Label: Node Id
Type: Int
Value: 16
Short ID: n
Parameter
ID: rx_gain
Label: RX Gain
Type: Float
Value: 0
Short ID: g
Parameter
ID: init_sync
Label: Initialise sync
Type: Int
Value: 0
Short ID: i
Parameter
ID: tx_nodes
Label: Tx nodes
Type: String
Value: 0
Short ID: t
Parameter
ID: rx_nodes
Label: Rx nodes
Type: String
Value: 0
Short ID: r
Parameter
ID: burst_period
Label: Burst period
Type: Float
Value: 500m
Short ID: p
Import
Import: np
Python Module
ID: utils
Variable
ID: preamble_len
Value: 257
Variable
ID: rise
Value: 17
Python Snippet
Section of Flowgraph: Main - After Init
Code Snippet: self...sleep(5)
Python Snippet
Section of Flowgraph: Main - After Start
Code Snippet: self..._alive()
ZMQ SUB Bus
Port: 5.6k
Timeout (ms): 100
Bind: False
Scheduler
Node Id: 16
Is RX: True
Initiate sync: False
ZMQ PUB Message Sink
Address: tcp://*:5600
Timeout (msec): 100
UHD: USRP Source
Sync: PC Clock on Next PPS
Start Time (seconds): 1
Mb0: Clock Source: External
Mb0: Time Source: External
Samp rate (Sps): 200k
Ch0: Center Freq (Hz): 680M
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Antenna: RX2
Ch0: Bandwidth (Hz): 200k
RX Sync (py)
Node Id: 16
Burst Tagger
True KeyID: Corr_max
True Value: True
False KeyID: burst
False Value: False
Measure tag delay
Burst length: 514
file name: record.csv
Delay
Delay: 257
Multiply Conjugate
Interpolating FIR Filter
Interpolation: 1
Taps: np.ones((preamble_len))
Complex to Mag^2
Peak Detector2
TH Factor Rise: 17
Look Ahead: 514
Alpha: 1m
Char To Short

ZMQ Bus


ZMQ SUB Message Source
Address: tcp://mnode1:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode2:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode3:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode4:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode5:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode6:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode7:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode8:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode9:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode10:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode11:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode12:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode13:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode14:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode15:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode16:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode17:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode18:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode19:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode20:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode21:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode22:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode23:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode24:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode25:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode26:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode27:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode28:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode29:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode30:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode31:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode32:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode33:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode34:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode35:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode36:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode37:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode38:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode39:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode40:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode41:5600
Timeout (msec): 100
out
ZMQ SUB Message Source
Address: tcp://mnode42:5600
Timeout (msec): 100
out
Pad Sink
Label: out
in

Signal detection process

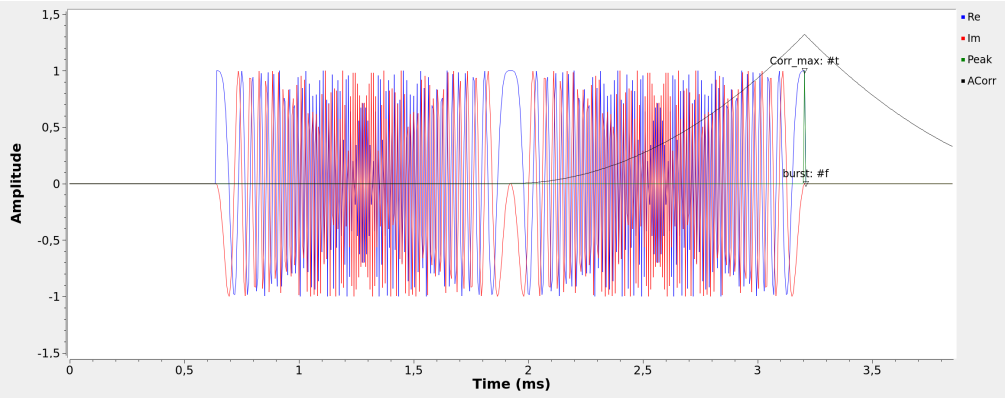
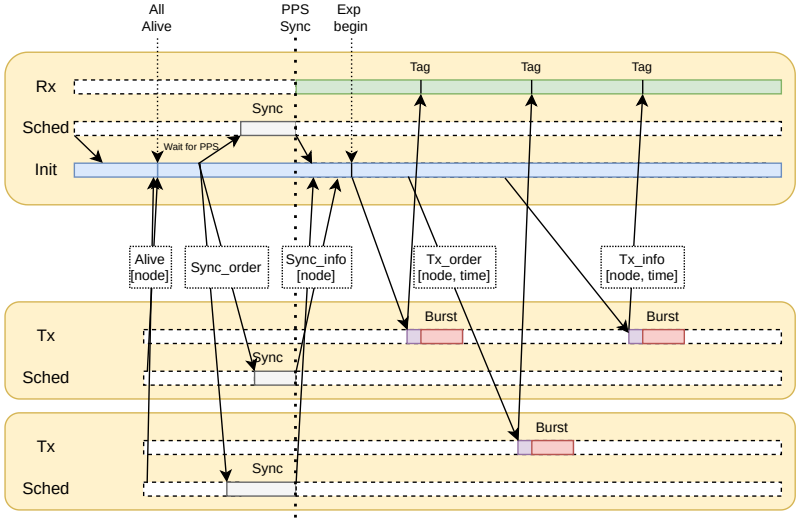


Figure: Repeated Zadoff-Chu sequence provides good autocorrellation peak at the reciever, matching the last transmitted sample Peak Detector2 and Burst tagger mark that peak in the input stream

Collaboration between nodes



Clock reset process

```
def do_sync(self):
    reset_trig_time = self.usrp_blk.get_time_now().get_real_secs()
    self.usrp_blk.set_time_next_pps(self.start_clock_value)
    self.clock_reset = False

    while not self.clock_reset:
        curr_usrp_time = self.usrp_blk.get_time_now().get_real_secs()
        if curr_usrp_time < reset_trig_time:
            self.log.info("Clock reset, start uhd source stream")
            self.clock_reset = True
            if self.is_rx:
                self.start_uhd_stream(curr_usrp_time)
            break
        time.sleep(0.1)

    info_msg = pmt.cons(pmt.to_pmt("sync_info"), pmt.to_pmt(self.node_number))
    self.message_port_pub(pmt.to_pmt("comm_out"), info_msg)^^I^^I
```

Sample time calibration

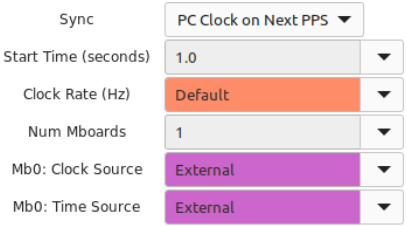


Figure: Starting USRP Source parameters

Process

- When clock reset, scheduler starts stream (as shown slide 10)
- Stream start triggers tag emission
- Rx sync block listens for rx_time tags
- Stores reference tag time and sample offset
- Counts offsets to measure time
- tx_info messages trigger tag emission on sample offset corresponding to tx time

Delay measurement

Measure tag delay block

- Knows preamble length
- Gathers sample rate from rx_rate tag
- Computes sample offset between expected time (+ preamble_len) and preamble correlation time
- Records transmitter, offset in samples and in seconds in csv file

Experiment automation

Modular flowgraphs

- Gets transmission elements from parameters
- Init scheduler gets list of transmitters and recievers from parameters

Generated scenarios

- Python script to generate yaml cortexlab scenario files from desired parameters
- Bash script to launch many, and gather results

Experiment design

Mix and match transmission-reception pairs

- Half room Tx, half room Rx, then switch
- All Rx receive simultaneously
- Cut room lengthwise, widthwise, and row-wise, to separate octoclocks

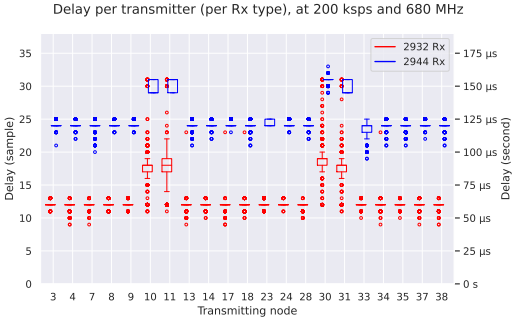
Transmission parameters

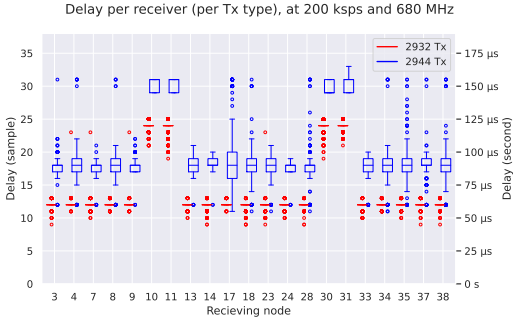
- Range of sample rates from 200ksps to 5Msps
- Couple of center frequencies with good transmission gains
- Adound 50 bursts per Tx, per type of cut, per transmission setting

What to look for

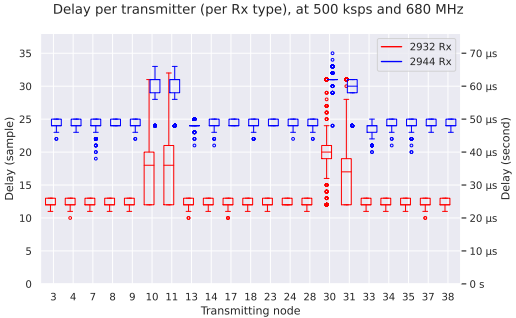
- Average delay between expected, and correlation time
- Delay spread

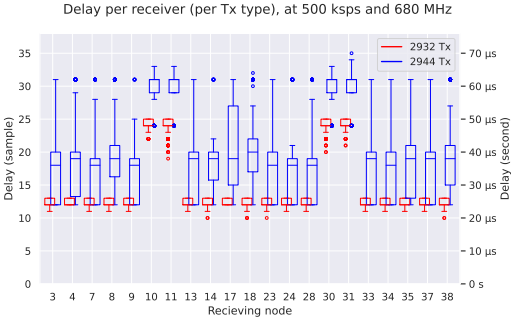
200 ksps, operating at 680MHz



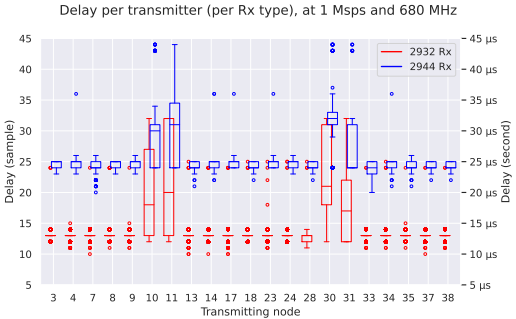


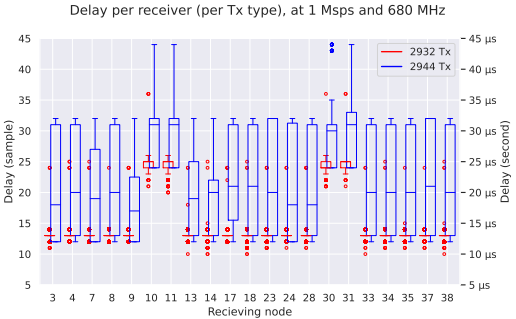
500 ksps, operating at 680MHz



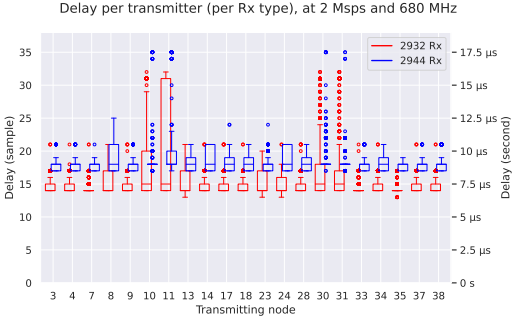


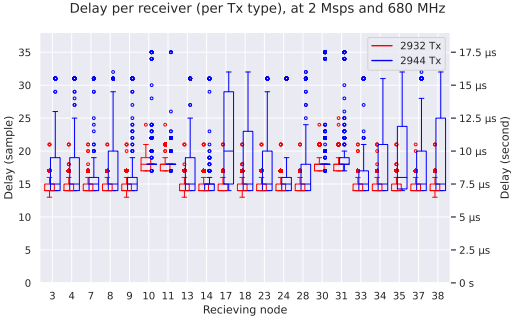
1 Msps, operating at 680MHz



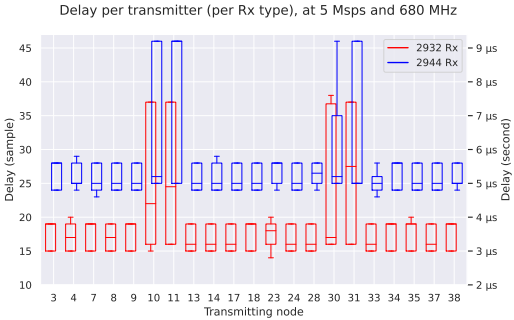


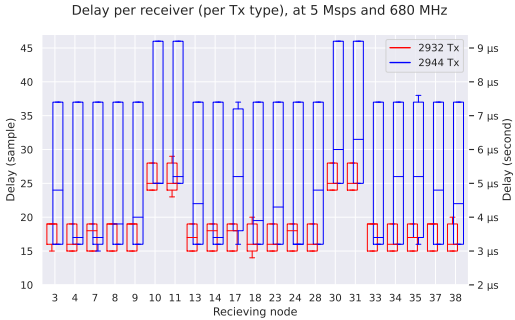
2 Msps, operating at 680MHz



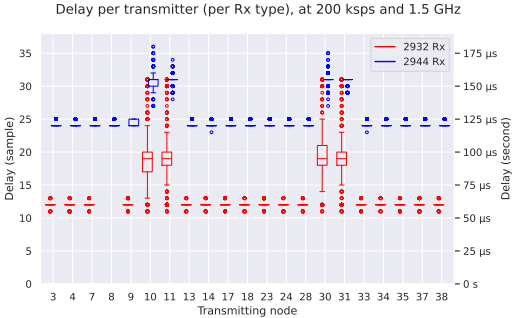


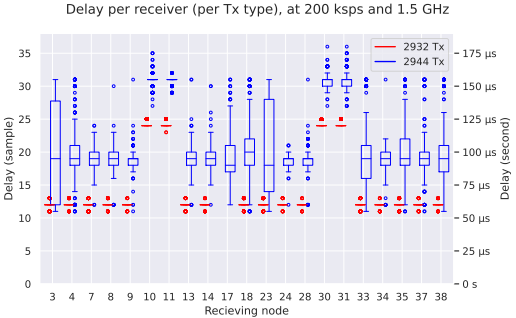
5 Msps, operating at 680MHz



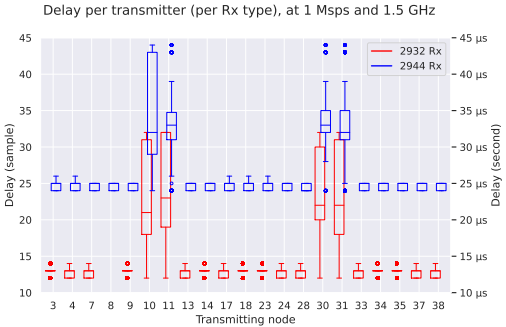


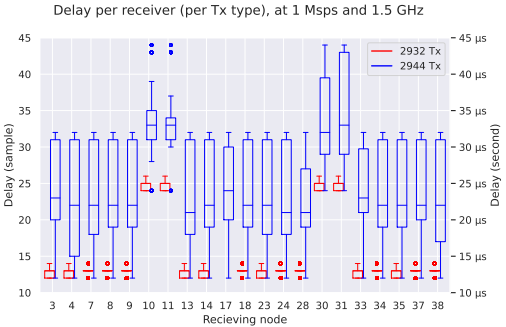
200 ksps, operating at 1.5GHz



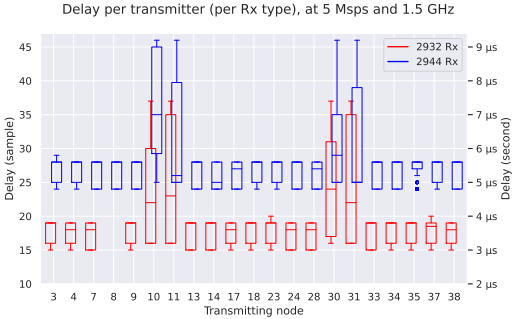


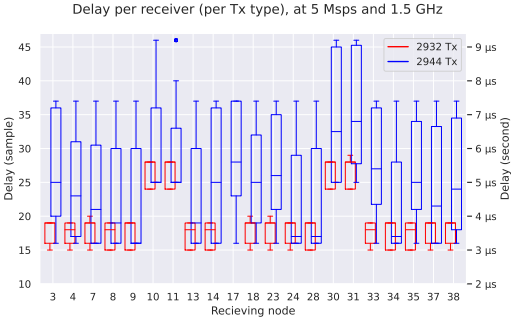
1 Msps, operating at 1.5GHz





5 Msps, operating at 1.5GHz





Conclusion

Reusable framework

- All coding done within python OOT blocks and GRC
- No messing with low level drivers and libraries
- ZMQ messaging handles flexible network setup

Synchronisation performance

- Offsets appear sample related, and not time based
- Higer sample rate gives higher absolute precision
- USRP 2932: reliable 12 sample offset
- USRP 2944: higher average and bigger spread, some dependance on sample rate
- Reminder, some spread attibutable to noise and Peak detection errors
- Once measured, delay could be compensated

Conclusion

Future works

- Sample rate difference between Tx and Tx
- Sweep more frequencies
- Operate at higher sample rates (With C++ blocks)
- Use as framework for time sync protocol evaluation (vs preamble, SNR, ...)

Reproducible results

- Code is available on gitlab: `https://gitlab.inria.fr/cortexlab/measurements/sync_evaluation`
- Public docker images
- Still requires some documentation
- Short term objective: convert to a CorteXlab tutorial/demo