Presentation of LatSeq

A Low-impact Internal Latency Measurement extension for OpenAirInterface, github.com/FlavienRJ/LatSeq

Flavien Ronteix–Jacquet

flavien.ronteixjacquet@orange.com

Introduction I

LatSeq for *Latency Sequence Analysis* is a tool developped by A. Ferrieux and F. Ronteix–Jacquet dM at Orange Labs Networks to analyse internal latency.
Internal latency is defined as the time between the moment the packet is fully received by the Base Station from an interface and the moment when all the segments making up the packet leave the software part of this node to be transmitted and possibly retransmitted.
OpenAirInterface (OAI) is an open source platform to develop LTE and 5G-NR Radio Access Network (RAN) and core [2][1][**?**][3].
An article is in proceedings describing philosophy of LatSeq [4].

Objectives

- Collect observed data packet fingerprint at different points onto their path inside node
- Rebuild data packet path from a list of fingerprints
- Analysis internal latencies of a system at a fine-grain

Usages

- Model Base Station as queuing system from inter-arrival time, service time and inter-departure time.
- Troubleshoot queues or scheduler behaviours

Introduction II

It is composed of **2 parts**, a *measurement* module and an *analysis* module. The architecture in figure below :

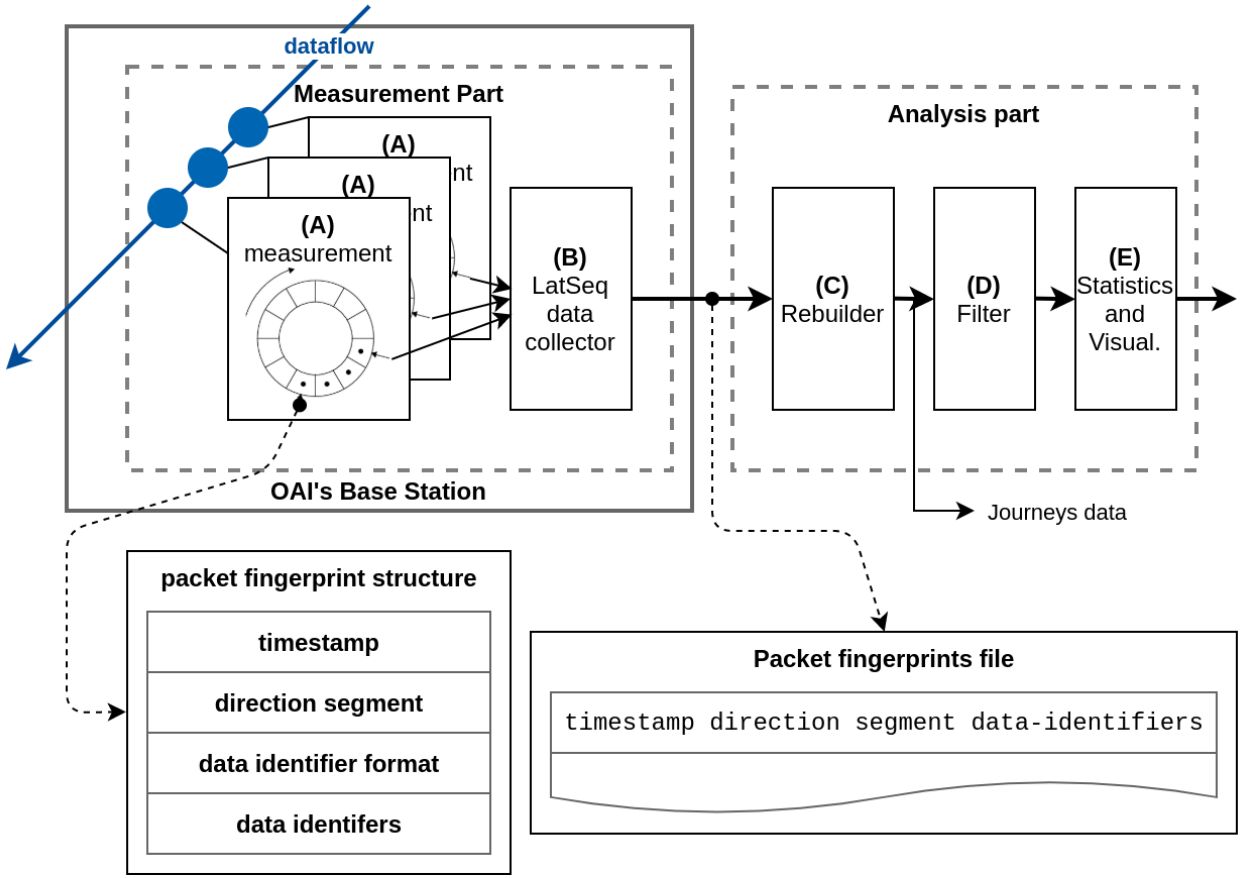


Figure – LatSeq tool architecture

The **normal usage** of LatSeq is as follow :

Introduction III

- put Latseq points `LATSEQ_P(protocol.ent.func, format_for_dataid, dataid)` into OAI Base Station code
- compile OAI BS with LatSeq enabled (`-enable-latseq`)
- run a **OAI RAN** on a scenario
- LatSeq points will log fingerprints into `/tmp/...`
- export **log.lseq** file
- parsing it with `latseq_logs` script
- rebuild packets' journey, possible paths,...
- filter logs with **latseq_filter**
- compute stats, analyze and display **visualization** (latency sequences, latency stats,...) from journeys

Measurement module I

LatSeq Architecture Figure1 shows that a part of LatSeq code is implemented in OAI source code. This code inside OAI code is composed of

- *latseq data collector thread* which is relatively independant from OAI (but must to be started by the main thread).
- *latseq points* (`LATSEQ_P(...)` inside OAI's functions. This part is really dependent of OAI Code... The knowledge of how OAI works is really important to have meaningfull measurements.

In source code :

- source directory for LatSeq measurement module : `common/UTILS/LATSEQ`
- source directory for LatSeq analysis module : `common/UTILS/LATSEQ/tools`
- build_oai flag : `-enable-latseq`
- macro enable : `LATSEQ`
- macro to log : `LATSEQ_P(point_identifier, data_identifier_format, data_identifiers...)`. *ex.* `LATSEQ_P("D pdcp.tx-rlc.tx.um","len%d:rnti%d:drb%d.psn%d.lcid%d.rsdu%d", ssize, ctxt_pP->rnti, rlc_p->rb_id, seqnum, rlc_p->channel_id, ssize);`

Measurement module II

How it works in OAI ?

- global logger_thread structure *g_latseq* (see Figure3)
  - flags : is_running, is_debug
  - init internals : filelog_name, time_zero, rdtsc_zero, FILE * outstream, latseq_stats
  - log_buffer : buffer of latseq_element to record, behave like a circular array
  - writer's and reader's positions in the buffer
- at the init, `init_latseq()` to call : init global latseq context and call `init_logger_latseq()` which init logger thread (see Figure2)
- the logger thread is a C's pthread which has the task to write elements from the buffer into log file. This needs to be in a different process, in background to avoid undesirable latency in logging. It is running until is_running flag is not 0 or until there is an error in writing files or until the log buffer is full.
- `LATSEQ_P(...)` is a macro which calls `_write_latseq_entry()`. This function try to add fastly a new measurement in the log_buffer. Time contraint on this function is huge because it runs in the oai threads.
- The buffer is a circular buffer which means, the writer's head should always be in advance of reader's head. If the buffer is almost full, we prefer to stop immediately latseq and print an error.

Measurement module III

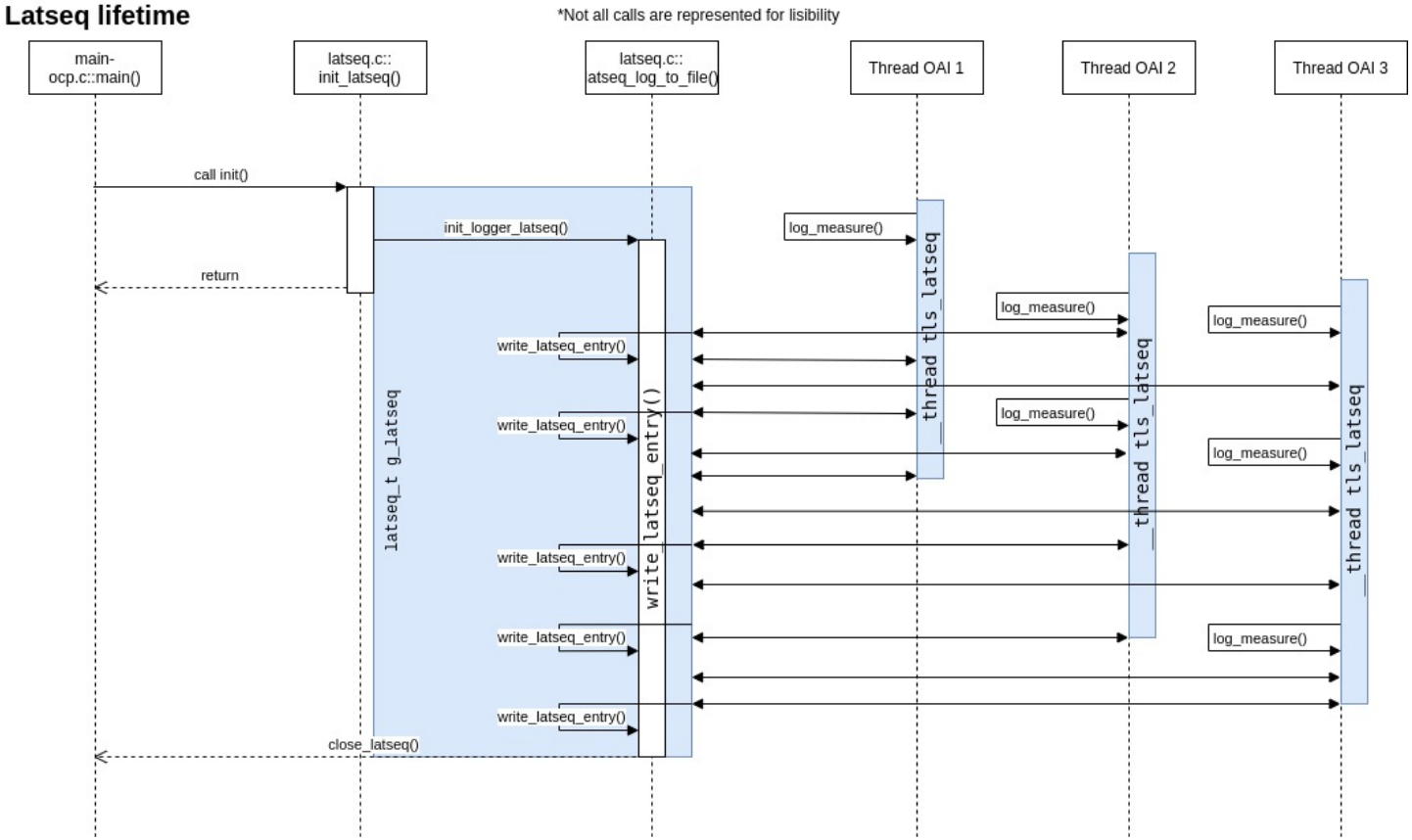


Figure – LatSeq Lifetime

Measurement module IV

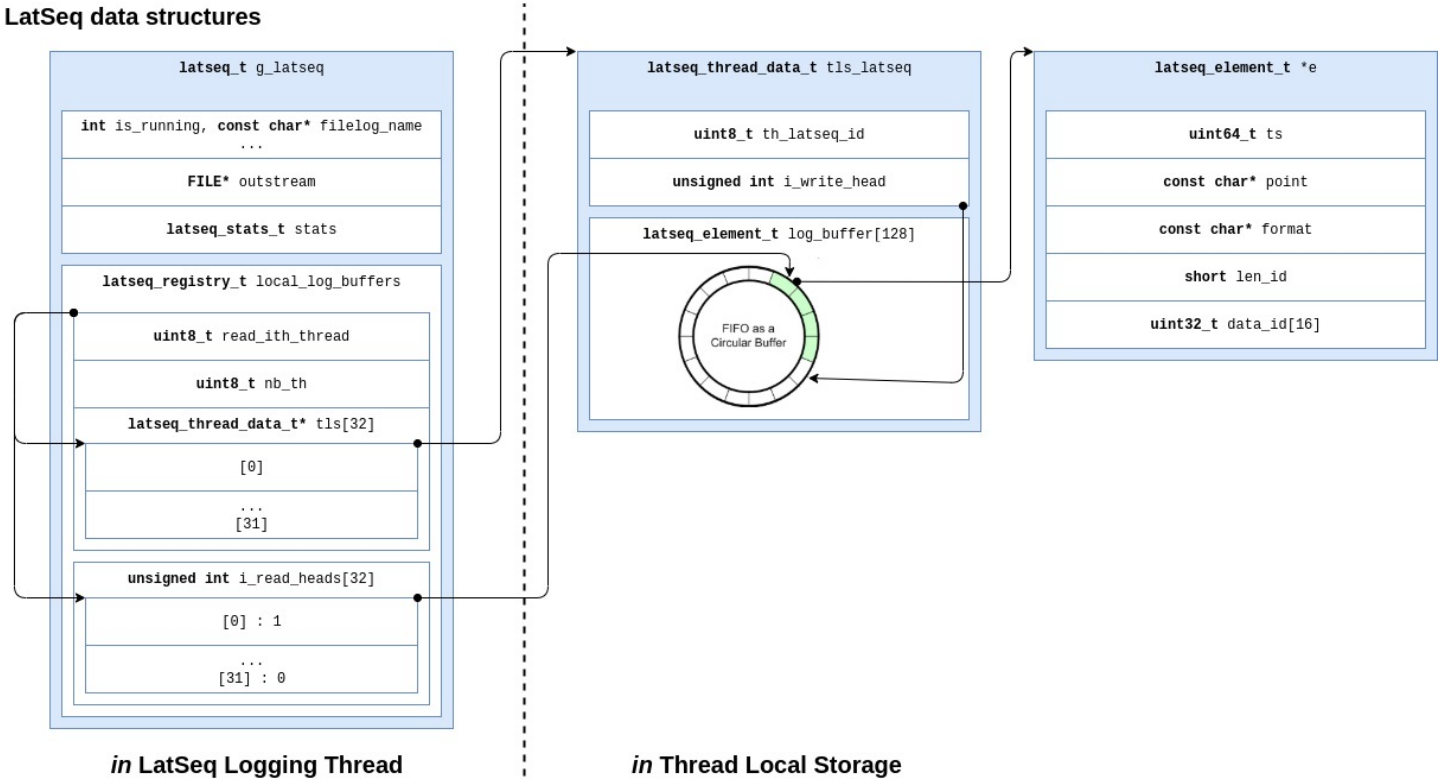


Figure – Latseq data structures

To ensure the objective of low-dependency, low-impact, low-memory requirements for measurements, these features has been implemented :

- Usage of `rdtsc` asm instruction to take timestamp
- Expansion, conversion and save of the entries **asynchronously** by the logger_thread and not by the oai_thread which performs the measurement
- Use of **inline function** for log_measure()

Measurement module V

- Use of **thread-local storage** (TLS) for thread-local buffer for measures instead of one big buffer in logger thread with the problem of mutexes, locks and waits. Thread local storage allows to reserve memory for LatSeq for all threads with a measurement point.
- Use of **lock-less ring buffer** for measurements.
- **Parametric Macro** instead of variadic function for log_measure. For 1 to 10 data identifiers, only one macro to use, `LATSEQ_P(...)`. We try to use a variadic function to have only one function, however, after test, this solution introduce a big overhead on instruction (11 to 19ns for 1 identifier, 16.3ns to 61ns for 10 identifiers).
- use of **const char*** for latseq point's name in LATSEQ_P. The copy takes only few asm instructions to be performed. Most efficient than allocates memory and free it later.
- Use of **fprintf()** for swift write of logging in file (with its buffering property)

Additionnal notes :

Measurement module VI

- The ring buffers size are set to **32** latseq logs elements. The buffer is filled by the LATSEQ_P points of the thread. If the thread has several Latseq measurement points, then this point log in the same ring buffer. Ring buffers are emptied by the logger_thread. Logger_thread visit all ring buffer of registered threads in turn. If the selectionned buffer is empty, then go to the next directly. By this, the most active ring buffer are emptied regularly. The ring buffer has to be sized according to the most active thread (in term of measurements). By experimentations, the speed of emptying is mostly conditionned by the disk speed. Then, the speed of measurement should not be to large of disk speed for a too long time. Buffers absorb this differences and absorb the time to visit all thread's buffers.
- To ensure we can rebuild packet's path and perform statistics and so on, at least one data identifer should link 2 measurements points. More local data identifier is given, less there are ambiguity.
- to ensure the journeys are rebuildable, the measurement points installer ought to be careful with : 1) the existence of a path between an In and an Out point (path existence property), 2) the global identifier never changes through the path of one journey (unique global identifier property), 3) at least one local identifier links 2 connected measurement points (the continuity property), 4) avoir loop between uplink and downlink's paths (strict direction seperation property)

Measurement module VII

the measurement points installer ought to be careful with :

- The existence of a path between an *Input* and *Output* point (*Path existence property*).
- The global identifier is consistent throughout the journey's path (*Unique global identifier property*).
- At least one local identifier links 2 connected points (*The continuity property*).
- No bridges between Uplink and Downlink's paths (*Strict directions separation property*).

These conditions enable to rebuild packet path as graphs which are :

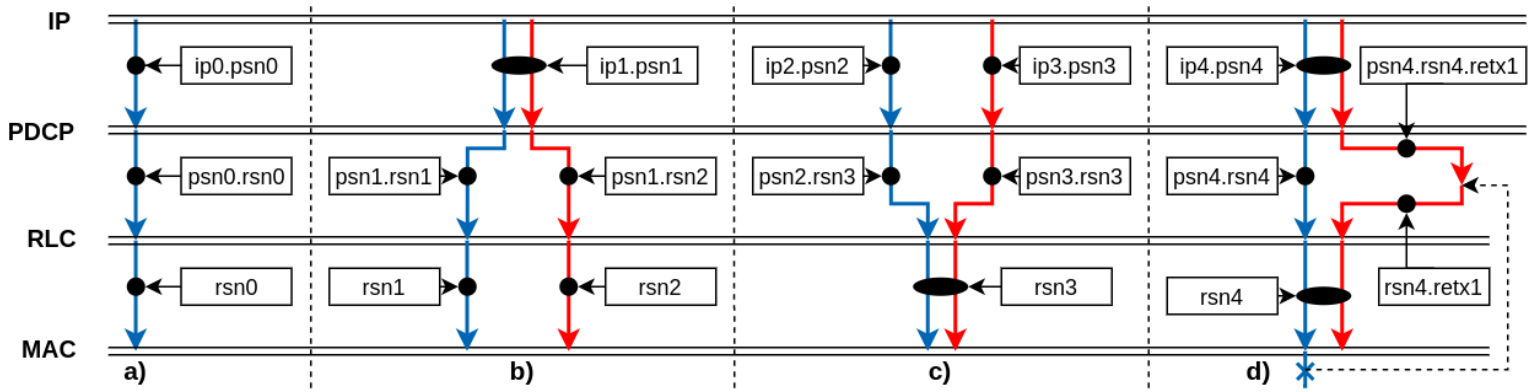


Figure – Cases : a) linear ; b) segmentation ; c) concatenation ; d) retransmission ;

Fingerprints

The link between on-fly measurements and analysis module is the trace file **.lseq* containing a list of fingerprints. A fingerprints represents an observed data packet at a point. The fingerprints is composed of :

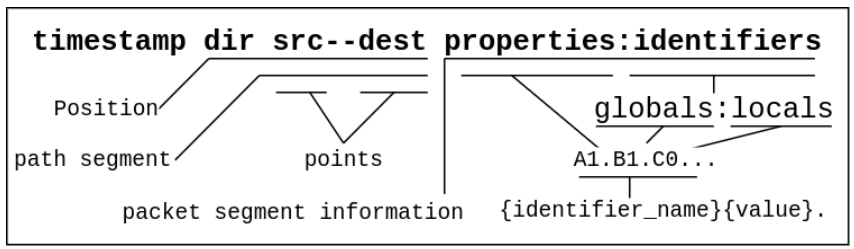


Figure – Fingerprints format

An example of trace file is given in Figure 10

Analysis Module I

How the post-processing and analysis module works ?

- from lseq logs, extract and clean measurements
- rebuild paths with an algorithm
- **latseq_logs** script could be called with this options (takes *.lseq in input and return *.json) :
  - -i : returns the inputs after filtering and cleaning as a string
  - -r : returns the paths present in the log file as json.
  - -p : returns points structure as json. (Becareful, if journeys has not been rebuilt, then you do not have "duration" attibute which is used for statistics)
  - -j : returns journeys structure as json. This is the main options to have statistics on packet's latency
  - -o : returns a latseq journey file line by line. redirects output to a file to have a *.lseqj and build "waterfall" visualization
- filter with **latseq_filter** to get statistics on specific category. for instances only downlinks points or packets with drb==1.
- request statistics to **latseq_stats** with options : (takes json in input and returns json or text) :
  - -j : returns statistics on journeys
  - -p : returns statistics on points

Analysis Module II

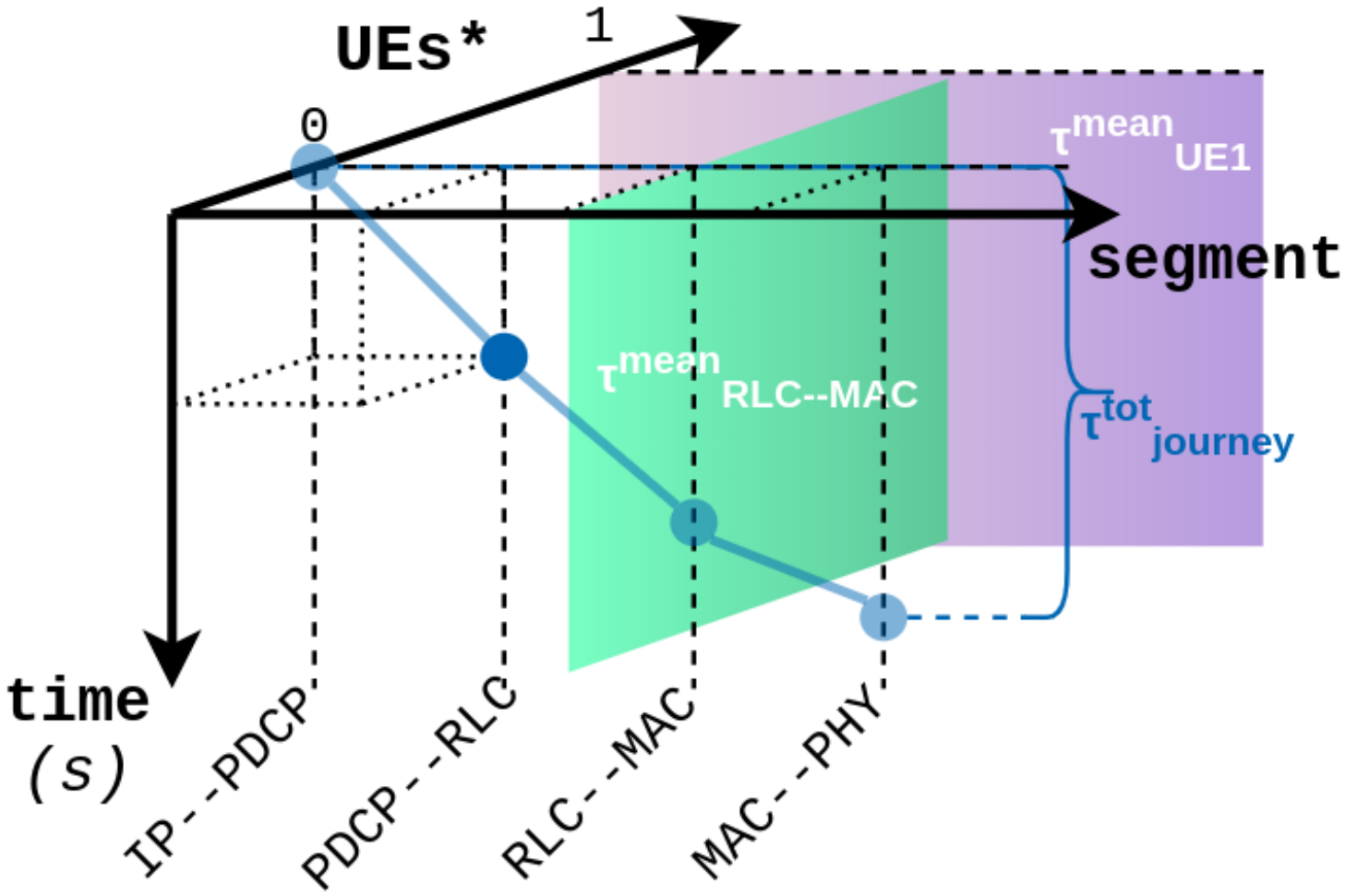


Figure – Matrix of delays

Analysis Module III

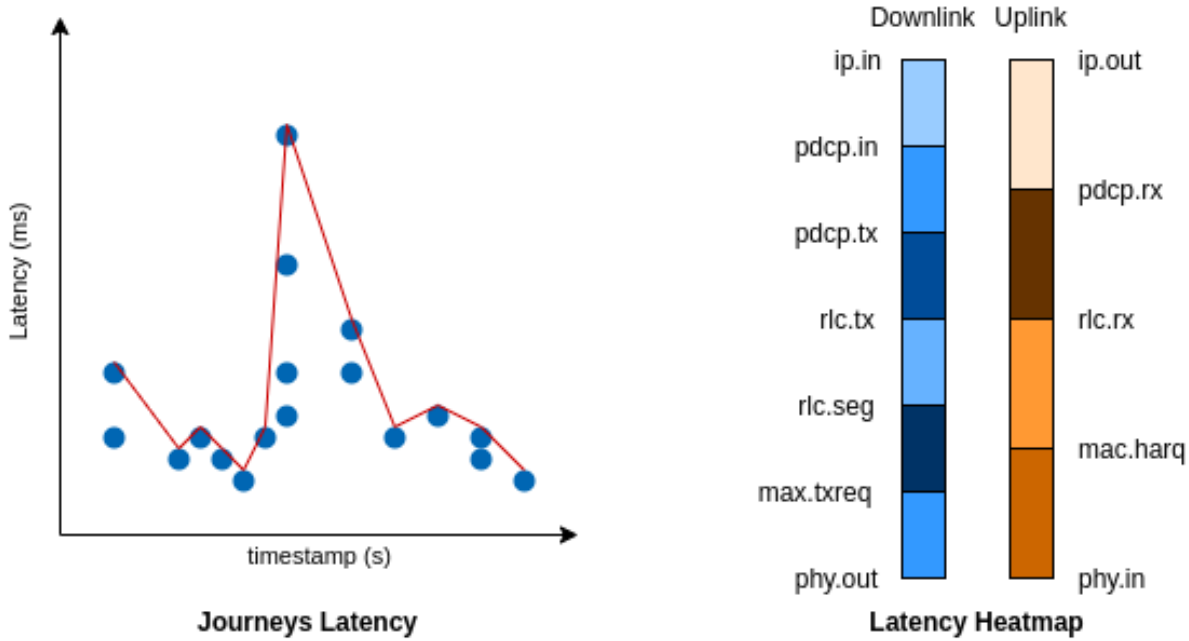


Figure – 2 differents representation of latency from matrix of delays

Analysis Module IV

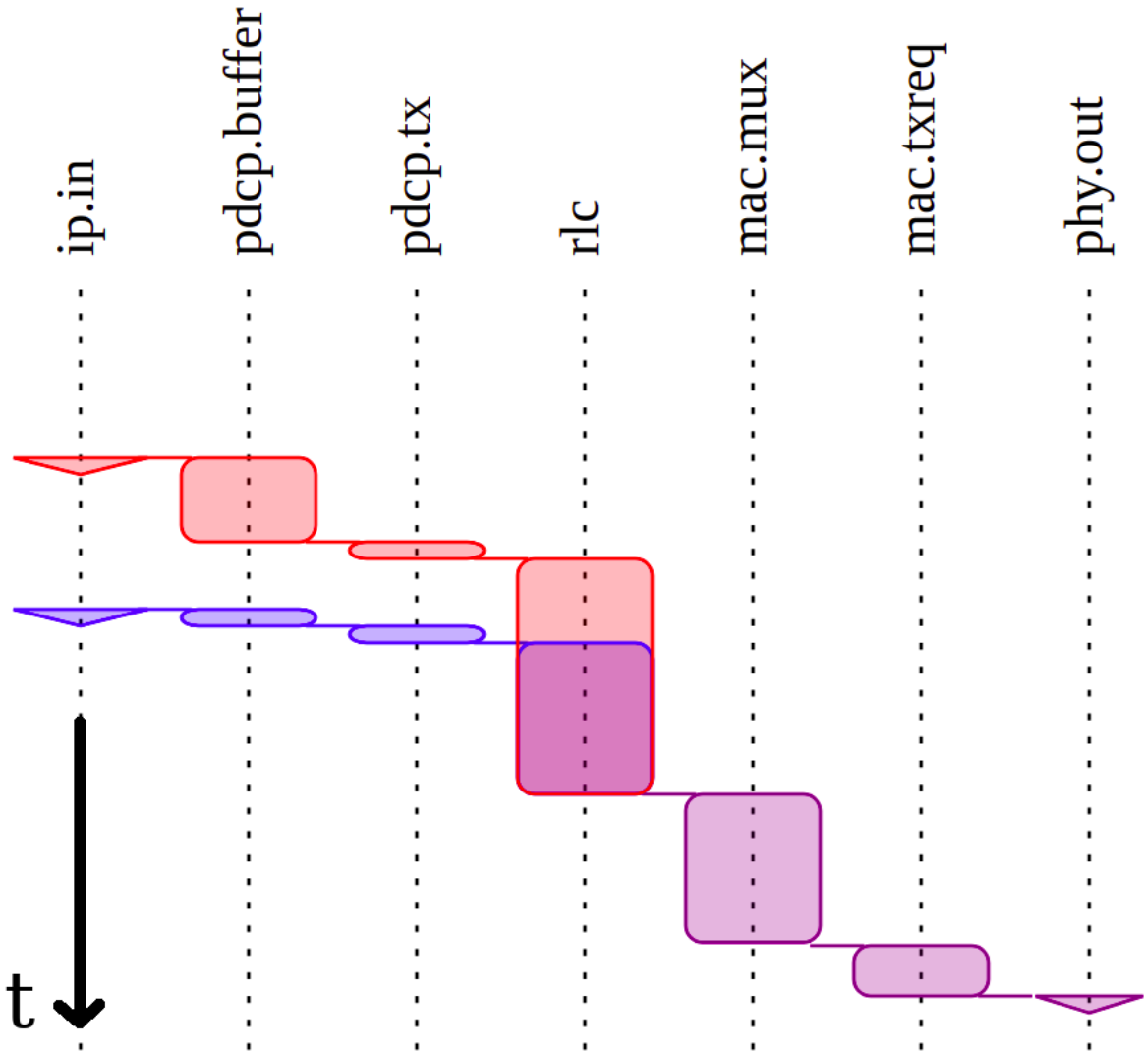


Figure – Waterfall of delays

Example I

An example of LatSeq implementation for OAI LTE's eNB.

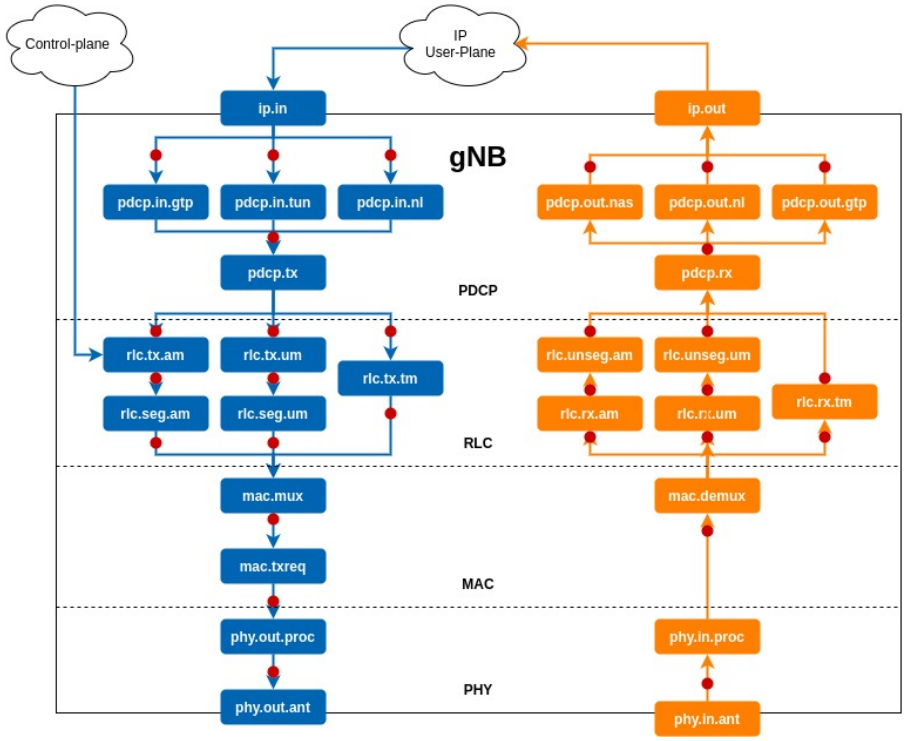


Figure – Latseq data points in OAI's eNB source code

Example II

```
1587645146.191801 D ip--pdcp.in.gtp len64:rnti54614:drb1.gsn12
1587645146.191802 D pdcp.in--pdcp.tx len64:rnti54614:drb1.gsn12.psn10
1587645146.191803 D pdcp.tx--rlc.tx.um len66:rnti54614:drb1.psn10.lcid3.rsdu0
1587645146.191962 D rlc.tx.um--rlc.seg.um len66:rnti54614:drb1.lcid3.rsdu0.rsn7.rso0
1587645146.191962 D rlc.seg.um--mac.mux len68:rnti54614:drb1.lcid3.rsn7.fm899
1587645146.191965 D mac.mux--mac.txreq len73:rnti54614:lcid3.harq7.txreq0.fm899.subfm1
1587645146.191997 D mac.txreq--phy.out.proc len73:rnti54614:ue2.harq7.fm899.subfm1
1587645146.207988 U phy.in.proc--mac.demux len60:rnti54614:lcid3.ue0.fm900
1587645146.207989 U mac.demux--rlc.rx.um len60:rnti54614:drb1.lcid3.rsn16.fm900
1587645146.207989 U rlc.rx.um--rlc.unseg.um len58:rnti54614:drb1.lcid3.rsn16.fm900
1587645146.207990 U rlc.unseg.um--pdcp.rx len54:rnti54614:drb1.lcid3.rsn16.psn0.psn12.fm900
1587645146.207994 U pdcp.rx--ip len52:rnti54614:drb1.psn12.fm900
```

Figure – Example of Latseq Logs at the output of Measurement module

Example of scripts pipeline

```
> ./latseq_logs.py -p -l ocp-softmodem.lseq | \
./latseq_filter.sh points_downlinks.lfilter | \
./latseq_stats.py -sp -P
> ./latseq_logs.py -j -l ocp-softmodem.lseq | \
./latseq_stats.py -f csv -djd > durations.csv | \
./gnuplot -e "data='durations.csv'" journeys_latency_boxplot.gp
```

Example III

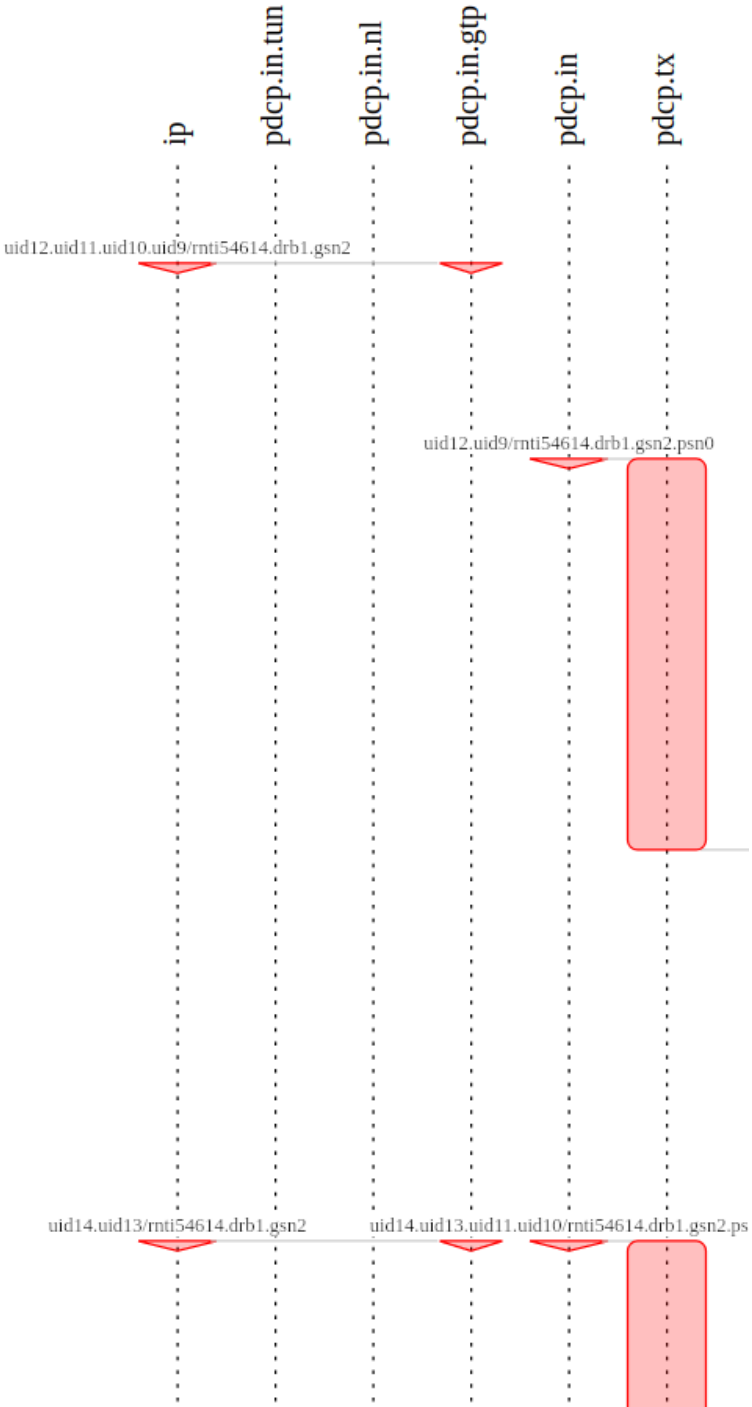
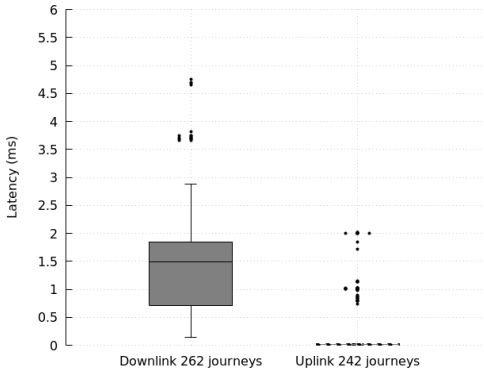


Figure – Waterfall visualization of data's journey

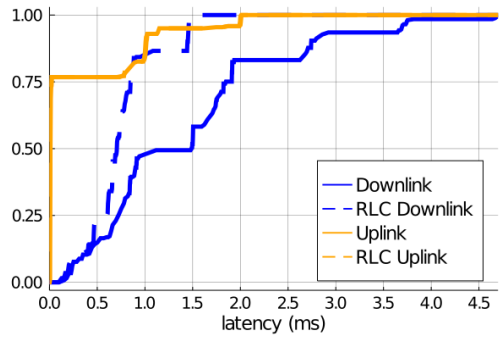
```
Paths found in /home/flavien/latseq.23042020.lseq
Downlink paths
        path 0 : ip -> pdcp.in -> pdcp.tx -> rlc.tx.um -> rlc.seg.um -> mac.mux -> mac.t
        path 1 : rlc.tx.am -> rlc.seg.am -> mac.mux -> mac.txreq -> phy.out.proc
Uplink paths
        path 2 : phy.in.proc -> mac.demux -> rlc.rx.um -> rlc.unseg.um -> pdcp.rx -> ip

Stats for Journeys latency
Values          |       Downlink
------          |       --------
Size            |       51715
Average         |       2.58
StDev           |       1.07
Max             |       5.62
[75..90%]       |       3.88
[50..75%]       |       3.7
[25..50%]       |       2.81
[10..25%]       |       1.82
[0..10%]        |       0.996
Min             |       0.0341
Values          |       Uplink
------          |       ------
Size            |       900
Average         |       0.269
StDev           |       0.547
Max             |       2.04
[75..90%]       |       1.01
[50..75%]       |       0.0522
[25..50%]       |       0.01
[10..25%]       |       0.00691
[0..10%]        |       0.00596
Min             |       0.00477

Latency for points
Stats for Point Latency for mac.txreq
Values          |       Downlink
------          |       --------
Size            |       646
Average         |       0.0316
StDev           |       0.00356
Max             |       0.052
[75..90%]       |       0.0319
[50..75%]       |       0.031
[25..50%]       |       0.031
[10..25%]       |       0.03
[0..10%]        |       0.03
Min             |       0.03
```

Figure – Formatted latseq_stats output statistics

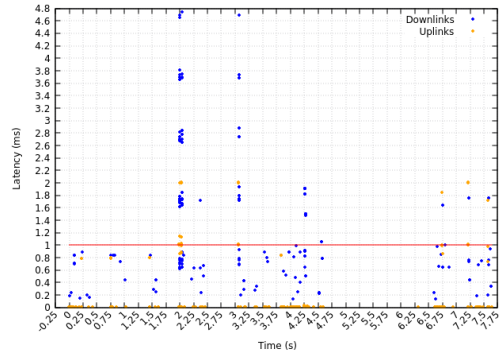
Scenarios



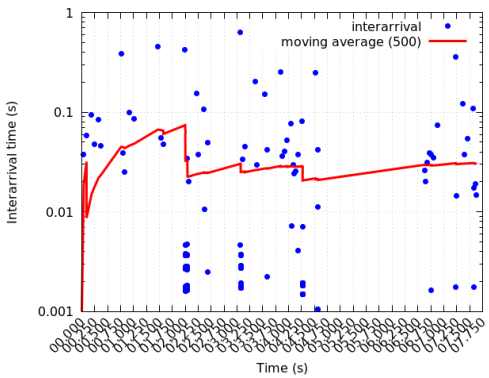
(a) Boxplot



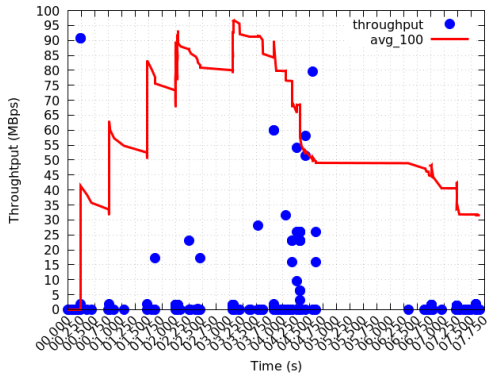
(b) cnorm



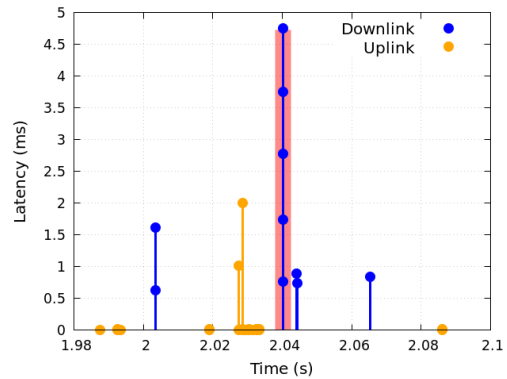
(c) dots latency for timestamp



(d) packet inter arrivals



(e) instant throughput at RLC



(f) latency dots cropped

Conclusion

Summary :

- Low-to-no-impact on observed system
- Reduced to minimum data collected of user data packet
- Easy-to-place LatSeq measurement points
- Access to packet journeys inside the system
- Allow fine-grain internal latency analysis

Limitations :

- Need to have access to source code
- Have a clear and comprehensive view on packet path throughout procedures and data structures to put significant points
- Only software internal latencis are measured, excluding input and ouput interface hardware latency

References I

[1] N. Nikaein, R. Knopp, F. Kaltenberger, L. Gauthier, C. Bonnet, D. Nussbaum, and R. Ghaddab.
Demo : Openairinterface : An open lte network in a pc.
In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom â14, page 305â308, New York, NY, USA, 2014. Association for Computing Machinery.
10.1145/2639108.2641745.

[2] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet.
Openairinterface : A flexible platform for 5g research.
ACM SIGCOMM Computer Communication Review, 44(5) :33â38, 2014.

[3] OSA.
Openairinteface 5g source code.

[4] F. Ronteix-Jacquet, X. Lagrange, I. Hamchaoui, A. Ferrieux, and S. Tuffin.
LatSeq : a Low-Impact internal latency measurement tool for OpenAirInterface.
In *2021 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2021)*, Nanjing, China, Mar. 2021.