LatSeq: A Low-Impact Internal Latency Measurement Tool for OpenAirInterface

Flavien Ronteix–Jacquet, Alexandre Ferrieux, Isabelle Hamchaoui, Stephane Tuffin, Xavier Lagrange

LatSeq: A Low-Impact Internal Latency Measurement Tool for OpenAirInterface

Flavien Ronteix–Jacquet*†, Alexandre Ferrieux*,
Isabelle Hamchaoui*, Stéphane Tuffin*, Xavier Lagrange†
IMT Atlantique, Rennes†
Orange Labs Networks, Lannion*
Emails: †{firstname.lastname}@imt-atlantique.fr, *{firstname.lastname}@orange.com

***Abstract*—Building a thorough understanding of latency in the 5G-NR RAN requires a reliable system level measurement tool able to collect relevant data in a broad range of situations.**

For this purpose, we propose *LatSeq*, an open-source software extension aimed at providing the OpenAirInterface Base Station with internal latency measurement capabilities. This paper discusses its design choices and evaluates its fitness for purpose in a first baseline usage scenario.

We demonstrate the low impact of *LatSeq* on the observed system, the usefulness of statistics it enables and the relevance of its capability of tracing individual packet journeys inside the base station.

***Index Terms*—Latency, Cellular Network, Measurements, Characterization, Open Source, OpenAirInterface.**

I. Introduction

Offering ultra low latency communications is undoubtedly one of the foremost 5G promises. However, coping with latency objectives under a few milliseconds creates tough challenges for the air interface design but also for QoS management and resource allocation on the radio segment.

The Third Generation Partnership Project (3GPP) designed the 5G New Radio (NR) interface with the aim of reducing the latency as far as possible using a lean design and flexibility [1]: slot duration to $1/8$ ms, resource allocation with symbol-level granularity ($1/14$ slot), delay between a data resource block and its acknowledgement lower than one slot. Such characteristics are a prerequisite to very low latency but they may be pointless without adequate scheduling algorithms, which remain vendor specific.

Many mechanisms have been proposed to cut down the latency [2], [3] including scheduling, active queue management and resource reservation. However, these mechanisms were generally not considered specifically for radio access. In [4], the authors review various mechanisms to reduce latency for wireline and wireless access. Moreover, a vast majority of the mechanisms proposed in the literature were only evaluated in simulated environments.

Furthermore, several open-source platforms of end-to-end mobile networks have recently emerged: OpenAirInterface (OAI) [5], [6] and srsLTE [7] are the most common ones. Both run on off-the-shelf hardware. The implementation of a 4G network is already available on OAI and the 5G version is under development unlike srsLTE. It is also supported by a relatively large and active community, which makes OAI a promising solution for research purposes.

Our objective is to develop a tool for the analysis of the latencies generated in a software Base Station (BS). Indeed, though each new generation dramatically increases its bit rate, the radio interface often remains a bottleneck. As a consequence, queues tend to build up in the BS, thus creating the delay we set out to minimize. In order to tackle these delays as a whole, one needs to investigate the complex interactions between the various layers at the radio interface.

The tool should then be able to measure the delays in each layer entity or more generally each process in which queueing or service time may build up. Delay values related to the same packet should not be captured independently from each other: what should be built is the sequence of successive latencies across the layers.

Making measurements inside a running BS is a challenge in itself as it requires to measure delays on the order of a microsecond for thousands of high throughput connections and hundreds of diverse users. Logging these large amounts of events necessarily incurs extra delays, while we are precisely seeking a low impact tool to preserve the BS operation.

To meet these harsh requirements, we designed a specific tool named *LatSeq*[1] for the analysis of latency sequences, and developed it over OAI. This tool measures, collects and displays various internal latencies and makes visible and understandable the life of data packets inside the BS.

This paper presents the design choices of LatSeq together with a preliminary performance evaluation. Section II introduces the notions of internal latency, journey and data structures. Section III presents the implementation of LatSeq and its key design principles. In the same section, LatSeq output designed for fine-grained analysis is also described. The low-impact aspect and performance assessment are shown in section IV. In section V we conclude and give some perspectives.

[1] https://github.com/FlavienRJ/LatSeq

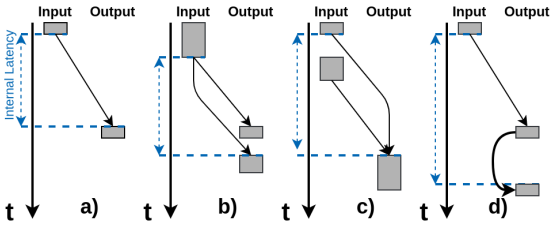


Figure 1: Latency and different possible operations on a packet: *a) linear; b) segmentation; c) concatenation; d) retransmission*

II. On Internal latency

A. Definition of internal latency and issues herewith associated

Defining the latency of a packet handled through a radio interface is in itself difficult. A packet is processed by a collection of layer entities and is prone to segmentation or reassembly, concatenation and retransmission as shown in Fig. 1. The latency can be considered either for each packet segment or for all segments related to the same packet.

Moreover, depending on the point of view (from inside the node, from outside the node, from software, from hardware), all the delays introduced are not accessible and do not have the same nature.

In this paper, we define the (e.g. downlink) *internal latency* as the time between the moment when the packet is fully received by the node (here the BS) from the wireline 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 (see Fig. 1). With this definition, the *internal latency* can be lower than 1 ms for 4G. It can be due to queueing, segmentation, retransmission, etc. in different layers of the protocol stack.

A second challenge is to track a packet or a data segment along its journey inside the BS. In the following, we call *data unit* a packet or a data segment in any layer. For both 4G and 5G radio interface, the protocol stack includes the physical layer, Medium Access Control (MAC), Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP). Each layer manages some local identifiers. For example, each User Equipment (UE) is identified at the MAC layer by the Radio Network Temporary Identifier (RNTI). Several flows can be transmitted to the same UE by use of different Data Radio Bearers (DRBs). At the RLC level, each data unit is referenced by a Sequence Number (SN) and several logical channels can be multiplexed and identified by a Logical Channel IDentifier (LCID). At the PDCP level, the data unit is also numbered with the PDCP Sequence Number (PSN). A single layer-dependent identifier (*e.g. LCID, SN, etc.*) does not provide a unique identification of a data unit. However, aggregating the identifiers from different layers makes it possible to track a data unit along its path. Note that some identifiers can be redundant (*e.g. the DRB number and the LCID*) but this does not present a problem.

B. The packet fingerprint structure

We propose to track a data unit during its life within the BS by creating *fingerprints* at different layers. A fingerprint is generated whenever a measurement is made. It includes a timestamp, a measurement point location and data identifiers.

The fingerprint format in the most general case is as follows:

```
t dir src--dest prop:globalId:localId
```

Where,

- `t` is a timestamp with a precision of one microsecond (standard Unix format for time);
- `dir` is the direction of transmission of the data unit;
- `src` and `dest` refer to the points (layer entity or the software procedure) between which the fingerprint is generated;
- `prop` includes one or several properties of the packet;
- `globalId` includes one or several global identifiers that remain constant along the life of the data unit within the node;
- `localId` is a list of local identifiers. The number of local identifiers and their nature may be different for the same data unit.

In the case of OAI, the format is

```
t dir src--dest length:rnti:A.B.C...
```

Field `dir` is either D for downlink or U for uplink. The property is just the length of the data unit; the global identifier is the RNTI; and the local identifiers (`A`, `B` and `C`) are typically indices or references at the different layer entities. An example of a fingerprint is:

```
146.191802 D pdcp.in--pdcp.tx
          len64:rnti513:drb1.psn10
```

This fingerprint means that at time $t = 146.192802$ s, a data unit on the downlink for UE with RNTI 513 is processed by the PDCP entity. This data unit is to be sent on data radio bearer 1 with PSN 10.

The next fingerprint of the same data unit is:

```
146.191803 D pdcp.tx--rlc.tx.um
          len66:rnti513:drb1.psn10.lcid3
```

We know that this second fingerprint is related to the same data unit thanks to common identifiers and matching source/destination fields, thus reflecting the continuity of the packet's life across layers.

C. Definition of packet journey

When launched, Latseq generates a *trace file* that includes a list of all fingerprints captured during its execution. We define the *journey* as the list of successive fingerprints for a given packet or a given packet segment. Due to segmentation and concatenation, a given fingerprint can be included in several journeys as illustrated by Fig. 2. Conversely, fingerprints related to different journeys are interleaved in the trace file; as a consequence, journeys are not distinguishable at

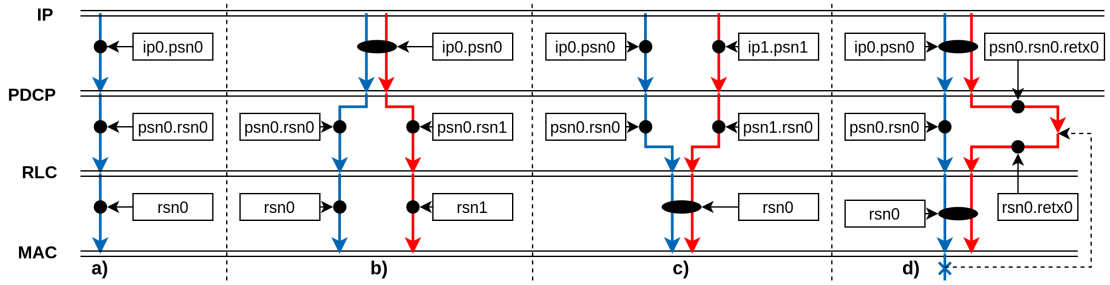


Figure 2: Graph of cases for journeys. a) linear; b) segmentation; c) concatenation; d) retransmission; *Black circles represent location of measurement points. Local identifiers to rebuild journeys are in rectangles*

the first glance and should be rebuilt thanks to appropriate treatments. Analyzing the journey of a packet informs about the processing at different layers and the associated durations. Collecting a set of journeys gives the capacity to study the latency at a fine grain.

D. Journeys as graphs

Each fingerprint can be seen as a vertex of a graph. When the `dest` field of a fingerprint is the same as the `src` of another one, there is an edge between the two vertices if both fingerprints share the same `globalID` (same RNTI in OAI) and the same `localID` values for each local identifier present.

In the trace file fingerprints related to different data units are interleaved; We are thus defining a graph that includes a lot of disjoint sets. Latseq generates a list of subgraphs, each one corresponding to the journey of a data unit. These subgraphs are directed and acyclic as shown in Fig. 2. All fingerprints generated at a given input share the same `src` field. For example, the fingerprints of all downlink packet arriving at the BS on the wired interface are characterized by `src=ip.in`. Similarly, when a data unit is transmitted on the radio interface, the `dest` field of its last fingerprint (i.e. the terminal node in the tree) is `dest=phy.out`. When a data unit is dropped by an intermediate layer, the terminal node is a fingerprint with a different `dest`.

Fig. 2 illustrates the 4 typical graphs, each one corresponding to a case of Fig. 1. Case d in Fig. 2 is particular because a packet that has already reached the output point reappears in the system due a retransmission.

III. Implementation

LatSeq is composed of two functional blocks as shown in Fig. 3:

- A *measurement part* : it captures fingerprints on a running BS and records them in a trace file. This part is itself made of a collection of *measurement points* inserted within the BS code, and a *data collector* module, which gathers all fingerprints and writes them to the trace file.
- An *analysis part* : it rebuilds packet journeys offline by scanning the trace file and computes data for analysis and visualization.

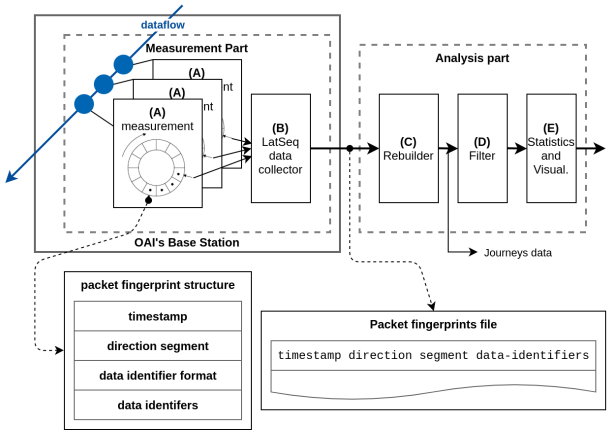


Figure 3: LatSeq tool's architecture.

The measurement part must have a limited impact on the execution time of the BS to avoid interfering with regular operations, whereas the analysis part has no strict time constraints.

A. Measurement points

Fingerprints are captured by inserting a macroinstruction called `LATSEQ_P` in the code. Whenever the macro `LATSEQ_P` is inserted in the code of an OAI thread, this thread is said to be *instrumented*.

Mechanisms	U	C
Variadic macro	✓	
Inlined measurement function	✓	
Time in number of CPU cycles (rdtsc)	✓	
Thread Local Storage (TLS)	✓	✓
Lockless ring buffer for temporary local storage		✓
Formatting by the data collector thread	✓	
Buffered IO	✓	

Table I: Implementation choices for LatSeq measurement module. *(U: for low CPU Usage, C: for low CPU Contention)*

The format of this macroinstruction is `LATSEQ_P("dir src--dest", "data_id1%d...", data_id1,...)`: the arguments contain all the necessary information for the fingerprint to be formatted in the trace file.

Because the macroinstruction handles a variable number of data identifiers, we use the variadic macro mechanism. The

timestamp is not given to `LATSEQ_P` as an argument, but computed internally by the quick assembly-level instruction "rdtsc" when capturing the fingerprint.

The macroinstruction code puts the fingerprint in a lockless ring buffer. This buffer uses static memory that is local to the instrumented thread: it uses the Thread-Local Storage (TLS) mechanism [8]. In other words, there are as many ring buffers as the number of instrumented threads.

Note that writing fingerprints in a non-thread-local buffer would have been highly inefficient, because concurrent thread access to such a buffer would have required inter-thread synchronization and thus would incur extra contention delays.

At this step, each fingerprint is stored in a ring buffer cell as a "packet fingerprint structure" (see Fig.3) for later formatting by the data collector.

This measurement method is inspired by [9], [10], [11], [12] solutions.

Note that OAI ships with T_tracer [13], a comprehensive logging tool for debugging purposes; however, we do not use it because of its limited logging performance.

B. Data collector

The data collector is a low-priority LatSeq-specific thread, low-priority so as to be conservative regarding Central Processing Unit (CPU) utilization and minimize its impact on the BS. This thread visits each ring buffer in a round robin fashion, formats the current fingerprint according to the structure described in II-B and writes it to the trace file using buffered IO. The trace file contains one line per observed fingerprint, providing fine-grained data for latency analysis.

The data collector is the only thread that empties the ring buffers and it runs asynchronously. If the size of a given ring buffer is not large enough, some fingerprints are lost. To avoid both overflow and waste of memory, a good trade-off should be found regarding the size of the ring buffers (see IV-A2). The implementation choices for both the measurement part and the data collector are summarized on Table I.

C. Overview of the rebuilder

The output of the LatSeq measurement part is the trace file including a succession of fingerprints related to different UEs, layers, packets, etc. The first task of the analysis module is to rebuild the journeys of the different packets. This is done by associating each journey with a graph as explained in II-D.

The trace file is sorted in ascending timestamp order. The rebuilder scans the trace file and looks for root vertices. Whenever a vertex is found, a child vertex is searched by scanning the remaining part of the file. This process is repeated until it reaches a vertex that corresponds to the transmission of the packet.

The result of the rebuilding phase is a list of journeys, each corresponding to a list of fingerprints.

D. The filter module

The LatSeq measurement part generates fingerprints for all data units forwarded by the BS. Should we want to restrain observation to a subset of them, for example to a given family of journeys (e.g. a flow), then a filtering module is required. Implementing this filter on the measurement side is out of reach, as the journey to which each fingerprint belongs to is calculated later, in the analysis part. The filter module should necessarily be implemented after the rebuilder.

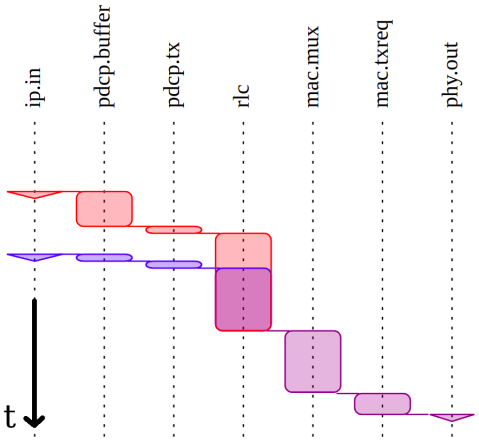


Figure 4: "Waterfall" representation.

With adequate filters in the analysis module, it is possible to investigate the behaviour of a protocol layer, a specific UE or a mechanism (e.g. segmentation). A wide range of filters can be defined. We give a few examples below.

- By selecting journeys with the same RNTI, the behavior of a specific UE can be studied.
- By additionally filtering on a local identifier (e.g. an IP packet number), the packet's entire life can be explained (see IV-D).
- The internal latency for a given bearer can be measured with a filter on a DRB value.
- The behaviour of any layer can also be analyzed with a filter on a `src` or `dest` value.

E. The statistics and visualisation module

We developed modules to compute statistics, which can then be visualised by standard tools. Moreover, we elaborated a visualisation tool specific to LatSeq that creates *waterfall* diagrams (Fig. 4). This type of diagram enlightens latency causality in a packet journey; it is described in IV-D.

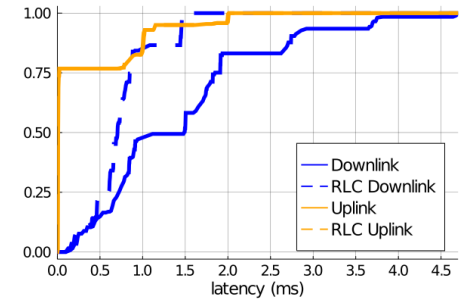
In this article, we focus on latency but LatSeq opens the door to a wide variety of analyses. For example, thanks to the `len` field, the instantaneous throughput can be computed, possibly at each layer or for each user. Extra information can be added in the `properties` field of the fingerprint to perform analyses based on other criteria.

IV. Use of LatSeq and evaluation

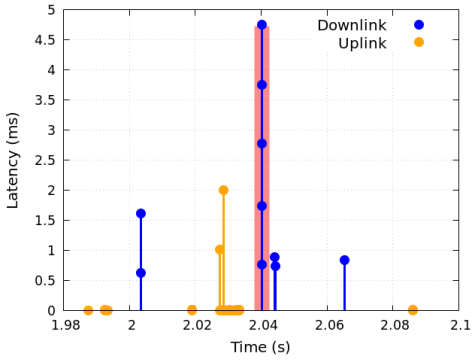
A. Impact of fingerprint generation and collection

1) Analytical evaluation of fingerprint generation: In a test environment[2], we inserted one LatSeq measurement point

[2]More details on https://github.com/FlavienRJ/LatSeq/test



(a)



(b)

Figure 5: Results from LatSeq; a) Cumulative distribution function of internal latency and latency due to RLC layer. b) Internal latency per arrival time.

Test	**without** LatSeq		**with** LatSeq	
	μ	σ	μ	σ
Fingerprint generation (ns)	0	0	14.13	0.273
event share of Data Collector (%)	0.0	0.0	0.21	0.03
CPU utilization by BS not loaded (%)	79	3	81	3
CPU utilization by BS loaded (%)	118	6.5	125	7.5
ping delay (ms)	4.46	0.27	4.8	0.53

Table II: LatSeq impact on BS performances results.

(with 5 arguments). From the decompiled code, we identified the LatSeq macro (see III-A) and (from the processor's datasheet) deduced its cost to be around 35 CPU cycles [14]. With a 2.60 GHz clock, this corresponds to a $35/2.60 = 13.5$ ns. Considering that 10 fingerprints are generated for a packet inside the BS, then LatSeq adds less than 200 ns of extra delay. We measure time with an accuracy of a microsecond, thus, the LatSeq extra-delay is 1/5 of the minimum quantum we measure.

2) Unitary performance evaluation of fingerprint generation: On the same simplified test environment, we measured the execution time of the LatSeq macro on 1 million runs from 1 to 10 arguments (*e.g. data identifiers*). The average time goes from 10.98 ns with a standard deviation 0.2 ns for 1 argument to 18.56 ns ($\sigma = 0.41$ ns) for 10. For 5 arguments, it is 14.13 ns ($\sigma = 0.273$ ns) which gives a 99% confidence interval of $[14.08, 14.19]$. This value is close to the predicted value but slightly different because of loop overhead.

3) Performance evaluation of the data collector: Remember that the collector asynchronously visits the various ring buffers in which fingerprints are temporarily stored, then formats them into a single trace file. To evaluate its contribution to LatSeq global performance, we measure its CPU consumption when fingerprints are generated at full speed (*i.e. max disk writing speed of 264 MBps in our setup*). On our simple testbed, we found that the collector thread is not CPU-bound (usage $< 100\%$). The data collector's capacity bottleneck is then clearly on the writing-to-file side, not on the fingerprint-formating one. We can then roughly estimate LatSeq's capacity as the maximum writing speed in ring buffers, that is about the full disk writing speed. At 264 MBps, for a fingerprint size of 100 bytes, it corresponds to 2.64M fingerprints per second. If we assume that each packet forwarded by the BS generates 10 fingerprints, then the maximum throughput we can measure accurately is $64 \times 2.64.10^6 * 8/10 = 136$ Mbps in the (worst) case of 64-byte packets. Moreover, as a running BS has no need for disk access, LatSeq does not threaten the system's performance even at full disk throughput.

B. Description of the testbed

The evaluation is made with OpenAirInterface LTE eNodeB installed on a computer with a CPU Intel Xeon E5-2640 at 2.60 GHz 16 cores, 16 GB of memory. The eNodeB is connected to an OAI Core Network (CN) and to Universal Software Radio Peripheral (USRP) B210 for the radio interface. The load is generated by one UE, which is an X52X from manufacturer LeEco. The transmission is made in Time Division Duplexing (TDD) with a 10-MHz bandwidth in LTE band 7 (2.6 GHz).

The UE sends a ping request every 20 ms to a server locally connected to the CN. Thus, the traffic is bidirectional with short packets.

C. Study of LatSeq impact with the OAI testbed

We instrumented the OAI code with 34 carefully chosen measurement points. Indeed, too many measurement points would be useless, as too few would make it impossible to rebuild journeys. On the downlink the input measurement point is set in the PDCP layer and the output one in the PHY layer. On the uplink, the input is set in MAC and the output in the PDCP interface with GPRS Tunneling Protocol (GTP).

Impact of LatSeq on the BS is summarized in table II. The `perf` test shows the CPU consumption of the data collector. CPU usage for a BS under light load focuses on the idle consumption of the data collector thread, whereas CPU usage under higher load shows the impact of data collector plus measurement points. A fingerprint element in the ring buffer has a size of 96 bytes. There is an order of 15 instrumented threads in OAI for one DRB. With this modest memory footprint, we can dimension the ring buffers under a few megabytes without any risk of overrun.

The ping delay denotes the (slight) impact of LatSeq induced by the extra CPU instructions. Results shows a limited

impact of LatSeq on experiments in view of the quantity and usefulness of the information it collects.

D. Studying the journey of a packet with LatSeq

The objective of the "*Waterfall*" (Fig. 4) representation is to show the journey of a packet through the protocol stack, as well as to enlighten the causality relationships between delays (who waits for whom ?). It is inspired by UML timeline diagrams and [15]. This representation of internal delays is useful to shed light on otherwise mysterious external observations.

Time is represented by the vertical axis. Boxes show the sojourn time in a layer or a procedure. The height of a box is proportional to the sojourn time. The box colour is assigned according to an identifier value (e.g. PSN). Concatenation, segmentation and retransmission are visible in the waterfall. In the example of Fig. 4, two RLC data units are concatenated by the MAC layer (MAC.mux in OAI) in one MAC data unit.

E. Analysis of the latency

Thanks to the filtering capability of the LatSeq analysis part, statistical analysis of specific latency components, e.g. for a given layer and/or for a given UE is possible.

Fig. 5a shows the Cumulative Distribution Function (CDF) of the total latency in the eNodeB for both uplink and downlink. It also shows the CDF of the internal latency contribution of the RLC layer. On the uplink, a transport block is generated every 1 ms by the physical layer and transmitted to the MAC layer, in which a measurement point is inserted. In about 75% of the cases, the transport block includes only one packet and the internal latency is very small because it is only due to processing. In the other cases, there is some reassembly and thus several data unit transmissions on the radio interface for the same packet; in this case, the latency can reach several ms. The RLC latency and the total latency CDFs are indistinguishable, which means the latency is only introduced by the RLC layer. On the downlink, packets arrive asynchronously on the wired interface but can only be sent every 1-ms. As a consequence, the cdf is more spread and shows some steps. The RLC layer generates a latency typically between 0.5 ms and 1.5 ms. Additional latencies between 1.5 and 4 ms are not due to RLC but to other layers since the CDF of the RLC latency is clearly separated from the CDF of the total latency.

CDFs as in Fig. 5a are useful to identify general trends. A deeper investigation is possible by representing the internal latency as a function of the packet arrival time as shown in Fig. 5b. Vertically-aligned points represent the latencies of a packet: when it has been segmented, each segment is transmitted separately and thus there are several associated latency values. For example, a segmentation in 5 data units is visible at $t = 2.04$ s in Fig. 5b.

V. Conclusion

In this article we propose LatSeq, a new tool to measure latencies in a software BS. This tool is composed of a low-impact measurement extension for the OpenAirInterface BS and a module to process and visualize the journeys of packets and various statistics. This paves the way to a very wide range of analyses of the latency at a fine grain. LatSeq provides for an understanding of interactions between the different layers in terms of delays. LatSeq can also be used to define and tune queueing and internal traffic models. This makes LatSeq a valuable help to optimise 4G and 5G radio interfaces.

The next step is to study in a more precise and comprehensive way latencies in OpenAirInterface5g BS to confirm and extend our first observations under different configurations. We also plan to fine-tune measurement point locations and improve the filter and visualization modules. It should be noted that LatSeq is sufficiently independent from the OAI platform's code and structure to be generalized to a wide variety of open-source software.

References

[1] S. Parkvall, E. Dahlman, A. Furuskar, and M. Frenne, "Nr: The new 5g radio access technology," *IEEE Communications Standards Magazine*, vol. 1, no. 4, pp. 24–30, 2017.

[2] M. Barreiros and P. Lundqvist, *QoS-Enabled networks: Tools and foundations*. John Wiley & Sons, 2015.

[3] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2149–2196, thirdquarter 2016, 6967689.

[4] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.

[5] 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*, ser. MobiCom '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 305–308. [Online]. Available: https://doi.org/10.1145/2639108.2641745

[6] 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*, vol. 44, no. 5, pp. 33–38, 2014.

[7] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srslte: an open-source platform for lte evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

[8] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[9] IBM, "Logging in multi-threaded applications efficiently with ring buffer," 2007. [Online]. Available: https://www.ibm.com/developerworks/aix/library/au-buffer/index.html

[10] Apache, "Log4j," 2014. [Online]. Available: https://logging.apache.org/log4j/2.x/misc/async.html

[11] H. Simpson, "zlog." [Online]. Available: https://github.com/HardySimpson/zlog

[12] A. Afanasyev, "Optimize multi-threaded logging using lock-free queue and separate thread," 2016. [Online]. Available: https://redmine.named-data.net/issues/2513

[13] OSA, "T tracer," 20XX. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/T

[14] A. Fog, "Instruction tables," 2019. [Online]. Available: https://www.agner.org/optimize/instruction_tables.pdf

[15] S. Mysore, B. Mazloom, B. Agrawal, and T. Sherwood, "Understanding and visualizing full systems with data flow tomography," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 2, pp. 211–221, 2008.