Gringoli, F., et al. Performance assessment of open software platforms for 5G prototyping, in *2018 IEEE wireless communications, 25(5), Oct. 2018, pp. 10-15*

DOI: https://doi.org/10.1109/MWC.2018.1800049

Performance Assessment of Open Software Platforms for 5G Prototyping

Francesco Gringoli, Paul Patras, Carlos Donato, Pablo Serrano, and Yan Grunenberger

***Abstract*—Given the urgency of standardizing the 5th generation mobile systems (5G) to meet the ever more stringent demands of new applications, the importance of field trials and experimentation cannot be overstated. Practical experimentation with cellular networks has been historically reserved exclusively to operators, primarily due to equipment costs and licensing constraints. The state of play is changing with the advent of open-source cellular stacks based on increasingly more affordable software defined radio (SDR) systems. Comprehensive understanding of the performance, limitations, and interoperability of these tools however lacks. In this article we fill this gap, by assessing by means of controlled experiments the performance of today's most popular open software Evolved Node B (eNB) solutions in combination with different commodity User Equipment (UE) and an SDR alternative, over a range of practical settings. Although these cannot underpin complete 5G systems yet, their development is progressing rapidly and researchers have employed them for 5G specific applications including LTE unlicensed and network slicing. We further shed light onto the perils of open tools and give configuration guidelines that can be used to deploy these solutions effectively. Our results quantify the throughput attainable with each stack, their resource consumption footprint, and their reliability and bootstrap times in view of automating experimentation. Lastly, we evaluate qualitatively the extensibility of the solutions considered.**

I. Introduction

The 3rd Generation Partnership Project (3GPP) has recently completed the specification of the 5th generation mobile systems (5G) architecture [1] and industry stakeholders are now focusing on finalizing the Release 16 documentation. Given the exceptional key performance indicators that 5G systems are expected to support, the need of field trials and experimentation driven by verticals is unprecedented.

Until recently, practical experimentation with cellular systems has been confined to the mobile operators community, primarily due to equipment costs and radio frequency licensing requirements imposed by regulators. In contrast, academics have gained vast experience by experimenting with technology that operates in unlicensed bands [2], such as IEEE 802.11 Wi-Fi and IEEE 802.15.4 ZigBee. Software Defined Radios (SDRs) have further facilitated the implementation of novel communications paradigms and medium access schemes (e.g., full-duplex Wi-Fi [3], cognitive radio transceivers [4], and white spaces networking [5]), but the complexity of the "higher layers" of the cellular protocol stack has precluded the development of complete systems by the academic community.

Francesco Gringoli is with the *University of Brescia*, Italy. Paul Patras is with the *University of Edinburgh*, UK. Carlos Donato is with *University of Antwerpen – imec*, Belgium. Pablo Serrano is with *University Carlos III of Madrid*, Spain. Yan Grunenberger is with *Telefonica I+D*, Spain.

However, multiple affordable "open" comprehensive software platforms have emerged over the past years [6], [7]. These put academics almost on par with their industry counterparts, as they are now able to rapidly implement full protocol stacks and evaluate new cellular technologies such as unlicensed LTE [7], [8]. By joining forces in practical experimentation efforts, potential exists to accelerate progress towards 5G standardization and keep up with end-user demands.

In this paper, we interconnect different platforms (including open software solutions and consumer-grade user equipment) to build end-to-end LTE systems, and assess their performance and adequacy towards 5G experimentation. We focus on two popular solutions that provide a complete version of the protocol stack, namely, OpenAirInterface (OAI) [6] and srsLTE [7].[1] OAI is licensed under the OAI Public License[2] and implements a subset of the LTE Release 10 specification, including key elements such as User Equipment (UE), Evolved Node B (eNB), Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SGW), and Packet Data Network Gateway (PGW), thus supporting a complete solution. OAI runs over commodity Linux-based computing equipment (Intel x86 PC architectures) and can be used with popular radio frequency (RF) frontend equipment (e.g., National Instruments PXIe and Ettus USRP). srsLTE is licensed under the GNU Affero General Public License[3] and also runs over a similar set of commodity equipment, providing implementations of the UE and eNB that are compliant with LTE Release 8. In addition to being compatible with commercial Core Network (CN) software solutions such as Amarisoft[4] or open-source alternatives including the OpenAir-CN, as we demonstrate in this paper, it has also recently released an implementation of the same CN elements.[5] Despite not constituting complete 5G systems, these platforms have been employed successfully for the development of several 5G technologies, including LTE unlicensed [7], [8], network slicing [9], neutral host deployments, or RAN sharing [10].

While there is certain *unwritten consensus* about the features of each of these platforms, namely, OAI being more computationally efficient than srsLTE, srsLTE's code base being more modular and easier to customize than OAI's,

[1] We do not include openLTE (http://openlte.sourceforge.net/) in our analysis due to its limited functionality (e.g., lack of a User Equipment software) and poor robustness, as compared with the other two solutions considered.

[2] http://www.openairinterface.org/?page_id=698

[3] https://github.com/srsLTE/srsLTE/blob/master/LICENSE

[4] https://www.amarisoft.com/

[5] We were not able to assess the performance with these elements, though, as they were not available by the time we ran our experiments.

there have been limited previous efforts to formally analyze their performance, or their interoperability. Existing work in this space only profiles the OAI components and measures its *emulation* execution time [11], or characterizes OAI's baseband processing times under a range of conditions [12]. In this paper, we set the record straight and carry out an extensive measurement campaign to characterize the performance of each platform under a variety of settings (including different UEs, channel bandwidths, and propagation conditions). We also analyze their interoperability, and discuss the degree of customization and extensibility they allow. We believe our results provide valuable insights and best practices to researchers and practitioners faced with deciding which platform(s) to consider for their experimental work.

The rest of this article is organized as follows. In the next section we describe our testbed, best practices for configuring the software, and the validation of our setup. Subsequently, we perform a quantitative comparison of the platforms in terms of throughput, CPU usage, network bootstrap time, and reliability. Finally, we discuss challenges facing platform extensions with custom features and give concluding remarks.

II. Testbed deployment, configuration, and validation

We first describe the hardware and software used in our testbed. Then, we detail the methodology we designed to set up the experiments. Lastly, we experimentally confirm that the performance observed in wireless scenarios is practically the same as then when coaxial cables are employed, which enables us to discard with confidence the impact of the transmission medium on the results.[6]

A. Testbed and tools

We conduct all the experiments reported using an eNB built with the Ettus USRP-B210 radio frontend boards connected using USB 3.0 to a Linux-based host computer that runs the different software suites considered. The frontend's up- and down-link interfaces are multiplexed on a single 2dB dipole antenna through a RF Diplexer compatible with LTE band 7. The host PC runs Ubunutu 16.04 and is equipped with an Intel Core i7-7700K CPU with four cores clocked at 4.2GHz, which is powered by an ASUS Z270-A motherboard and 16GB of DDR4 memory. We remark that a configuration with such high computing power is required to be able to run effortlessly the different software solutions that we use, since baseband processing is particularly CPU expensive.

Using the configuration described above, we conduct experiments with two types of eNB software, namely:

- **OAI** – version 0.6.1;
- **SRS** – version 2.0-17.09 of the srsENB application.[7]

We use an additional computer to host the Core Network (CN) functionality. It runs the OpenAir-CN stack, which implements an open-source Evolved Packet Core (EPC): MME, HSS, and P/S-GW modules. The computer providing eNB functionality through the different solutions considered is connect to the CN using a Gigabit Ethernet link. Although we could virtually run both the Radio Access Network (RAN) and CN stacks on the same physical equipment in order to reduce deployment costs, the configuration we adopt ensures the performance of the eNB will not be affected by the computing load placed by the processes specific to the CN.

In terms of UEs, we experiment with three different solutions: two of these are commercial of-the-shelf (COTS) equipment, and the third one is an SDR platform with open-source software commonly used for cellular testing purposes. Precisely, we work with an LG Nexus 5 smartphone (denoted 'Nexus' throughout our experiments), a Huawei e3272-153 USB dongle (denoted 'Huawei'), and respectively the Ettus USRP B210 board running the srsUE stack version 2.0-17.09 (denoted 'srsUE').[8] To control the Nexus 5, we connect it through its USB interface to a PC-Engines APU embedded computer,[9] which runs the Android Debug Bridge software[10] to access the internal command console of the smartphone. We use the same APU device with the Huawei dongle, which we supply with external power through an USB hub, to prevent power-related instability issues. To be able to test with the commercial equipment considered, we fit the smartphone and USB dongle with Sysmocom programmable SIM cards.[11]

During each experiment, only one UE is connected to the eNB. Furthermore, to avoid external interference on our experiments, as well as our experiments interfering with external networks, we placed our network setup within an RF shielding tent that is easy to set up and provides attenuation in the -50 to -60dB range.[12] Finally, to generate and receive traffic, we install `iperf`-compatible software on the Nexus 5, the APU device hosting the Huawei dongle, and the computer running as SDR UE. During our experiments, we transmit UDP traffic in the uplink or downlink directions, and collect statistics at the receiver by sampling `iperf`'s output every second. We also measure the total CPU usage of the SDR process at the eNB by executing the `ps` command and querying the corresponding process ID.

B. Experiments set up and repeatability

For each combination of equipment and software we first devise a procedure for determining the optimal configuration of the TX and RX gains at the eNB. This turned out to be required to ensure the resolution of the ADC/DAC is not compromised or the receiver does not saturate. This is because both implementations do not adjust the output power level of

[6] We note that we faced a number of performance issues (e.g., configuration-dependent incompatibilities, out-of-memory crashes) during our experiments, which was was somehow expected as both open projects are actively evolving. We report the most relevant ones, for the software versions available at the time of testing, in Section IV.

[7] The srsLTE software suite comprises an eNB stack (srsENB) and the UE stack (srsUE). Hereafter, whenever there is no scope for confusion, we use SRS and srsENB interchangeably to refer to the eNB software.

[8] Although an OAI UE implementation exits, we found that this was incompatible with the SRS eNB software. Therefore, for a fair comparison of the eNB stacks, we do not report the performance of an OAI (eNB, UE) pair.

[9] http://www.pcengines.ch/apu.htm

[10] https://developer.android.com/studio/command-line/adb.html

[11] http://shop.sysmocom.de/products/sysmousim-sjs1

[12] http://www.globalemc.co.uk/shielded-tents.php

the boards, but rather adjust signal sample amplitudes by scaling. The procedure consists of running very short experiments (e.g., UDP traffic sessions up to 5 seconds) and measuring the throughput obtained for each combination of gain. With these measurements we fill throughput matrices for the uplink (UL) and downlink (DL) directions, as shown in Fig. 1, which illustrates the case when the eNB runs srsENB over 10 MHz channels and the UE is the Huawei USB dongle connected with coaxial cables to the eNB. Subsequently, we employ the gain pair that yields the highest throughput. Depending on the direction of each experiment run, we may set different gain pairs. We note that the implementation of the OAI stack is more accurate, since in most cases setting these hardware gains to the maximum value led to highest throughput performance.

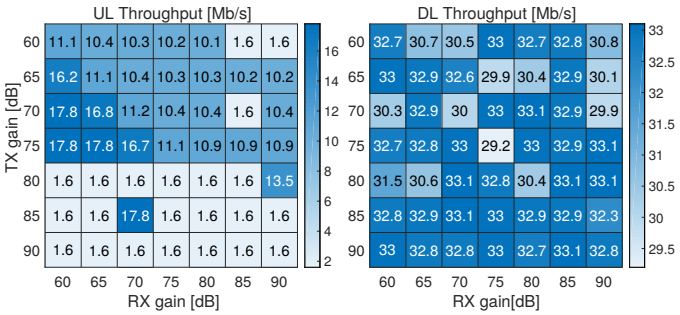

Fig. 1. Impact of eNB RX/TX gain on the throughput achievable in the uplink (left) and downlink (right) directions. The eNB employs the srsENB stack over a 10MHz channel, the UE is a Huawei USB dongle, and the propagation medium is coaxial cable.

Before each repetition of a conducted test, we completely restart the network and measure the time required to establish a working client connection, i.e., until the UE has Layer 3 connectivity with the CN, as verified with ICMP echo request/reply. On the one hand this enables us to build statistical significance for the performance metrics of interest. On the other hand, we also collect data about the bootstrap instances that did not conclude successfully, in order to report on the reliability of specific configurations (this will be done in Section III-D). As such, we bootstrap experiments through a set of `Bash` scripts that involve the following steps:

1) Starting the CN and verifying that all processes are working and remain alive;
2) Starting the eNB and waiting for connection to the HSS;
3) Starting the UE, waiting until this finds the cell, and initiating the attachment process;
4) Waiting until the UE obtains an IP address and receives an ICMP echo reply from the traffic generator running at the CN;
5) Measuring the achievable throughput by running five consecutive transmissions, each of 5 s duration.

After the network has been setup successfully, we run 30 s long transmission experiments, setting the offered close to the the maximum throughput measured at the last step of the bootstrap procedure.

C. Validation

To confirm that the proposed evaluation methodology is reliable and not susceptible to inaccuracies induced by practical signal attenuation and multipath propagation, we first compare the throughput attainable in both directions (UL and DL) over 5 and 10 MHz channels with all the possible SW/HW configurations, when the eNB and UE communicate over a wireless channel (air) and over coaxial cables using SMC connectors (cable), respectively. Note that our validation does not include experiments with the Nexus smartphone, as instrumenting wired connections would have voided the device's warranty. We illustrate the results of these preliminary experiments in Fig. 2.

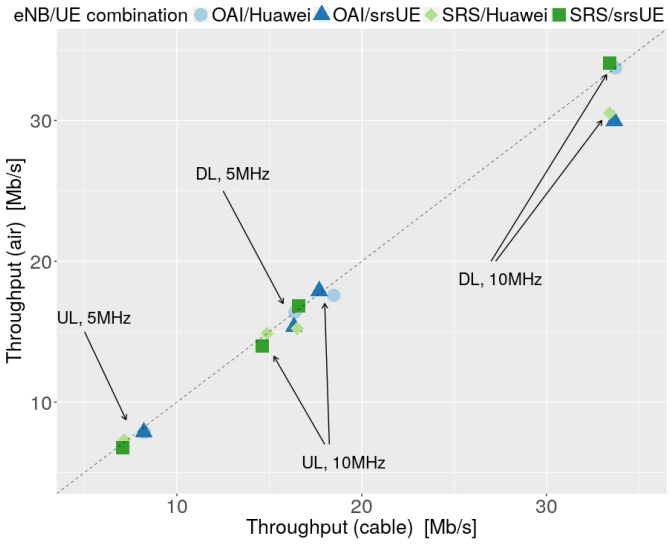

Fig. 2. Methodology validation: throughput performance over wireless vs. wired medium for different eNB/UE combinations, channel bandwidths, and transmission directions.

In this figure, we plot the throughput performance over the air vs. that obtained over the wired link. We investigate this for each direction, bandwidth, and eNB/UE configuration considered. Observe that for each combination the results are highly correlated, with a Pearson correlation coefficient of $r = 0.993$, which proves that the communication medium has practically no impact on the achievable performance. Therefore, we are confident that the performance differences, which we report in the next section for all the possible configurations, have other root.

III. Performance Evaluation

In this section, we compare the performance in terms of throughput, CPU consumption, and bootstrap time achievable with different eNB stacks (OAI vs srsENB), UEs (Nexus smartphone, Huawei USB dongle, and srsUE with USRP SDR), transmission directions (UL/DL), and bandwidth (5 and 10 MHz) configurations.

A. Throughput performance

We start our analysis by assessing the throughput performance when sending unidirectional UDP traffic at the maximum rate achievable in the uplink (from the UE) and downlink (from the eNB) directions. To this end, we use `iperf` to generate 1500 B frames from the corresponding side (the UE

in the former case, and the computer hosting the CN stack in the latter), and measure the average throughput obtained every second, during 30 s trials. We plot in Fig. 3 the average and 95-percent confidence intervals obtained following 20 repetitions of each test. In this figure each direction is represented with a different color, and each subplot corresponds to a different configuration of the eNB (OAI or SRS) and BW used (5 MHz or 10 MHz). To add perspective, we show with dashed lines the theoretical maximum throughput achievable in each case.

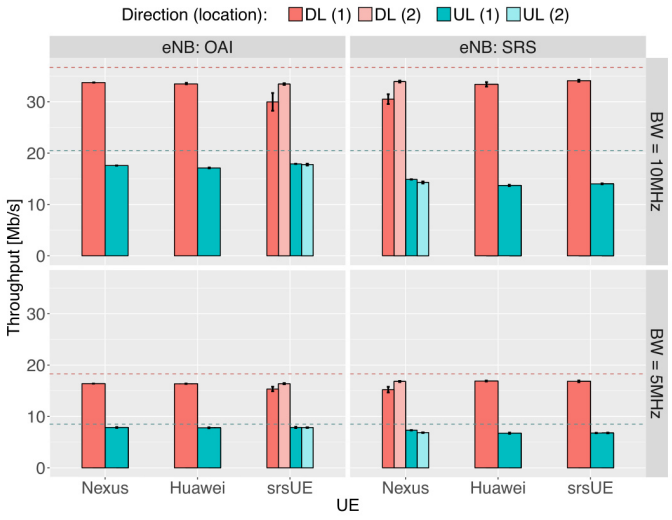

Fig. 3. Throughput performance obtained with different eNB/UE configurations, UL/DL directions, and 5/10 MHz bandwidths. Experiments repeated at a second location (2) with SRS, to illustrate the sensitivity of this stack to PHY conditions. Shown with dashed line is the theoretical maximum throughput in each case.

There are two key results worthwhile remarking in the figure, apart from the obvious performance asymmetry in terms of the UL/DL rates, which is due to the fact that the DL operates with 64QAM and by default the UE transmits using 16QAM. Firstly, the type of UE used has practically no impact on the performance attainable with a given eNB, since in all cases the obtained throughput is almost "flat." We only note minor differences between the commercial equipment's performance (i.e., Nexus and Huawei) and that of the experimental platform (srsUE). In particular, for the case of {DL, 10 MHz} (top left subplot) we note a slight drop in performance when srsUE is employed with OAI. We also note that initially the throughput attained with the (SRS, Nexus) combination was lower. Our intuition is that the signal processing performed by the SRS stack (both in the eNodeB and UE implementations which are built on the same library) is different from that of OAI. Specifically, in some scenarios an SRS receiver may underestimate the channel quality, or produce a stream of samples that cannot be correctly decoded by a commercial receiver. To confirm this we repeated these tests with the UE placed at a second location, where the performance of SRS was at the same leveled with that of OAI – note the lighter bars labeled with "(2)" in the figure. As the eNB and UE use the same PHY stack with SRS, any mismatch is less likely to occur.

Secondly, we note the UL rates are slightly smaller when the eNB runs the SRS stack instead of OAI, something that is particularly noticeable for the 10 MHz configuration. As we analyze next, this can be related to the CPU demanded by the srsENB solution when decoding traffic, particularly in the uplink direction. Despite these small variations, it is also worth remarking than SRS and OAI provide a similar throughput performance, with the difference over all cases being on average smaller than 7%.

B. CPU Usage

Next, we analyze the CPU usage of the cellular SW stack running at the eNB, aiming to quantify the differences in terms of resources consumed by the OAI and SRS solutions. For this purpose, we repeat the same experiments performed above, identifying the process ID corresponding to the stack of interest running at the eNB, then invoking the `ps` command every second, to estimate the CPU load. We illustrate the average and 95% confidence intervals of the CPU consumption for all combinations considered in Fig. 4, where the subplots are arranged as in the case of the previous experiments.

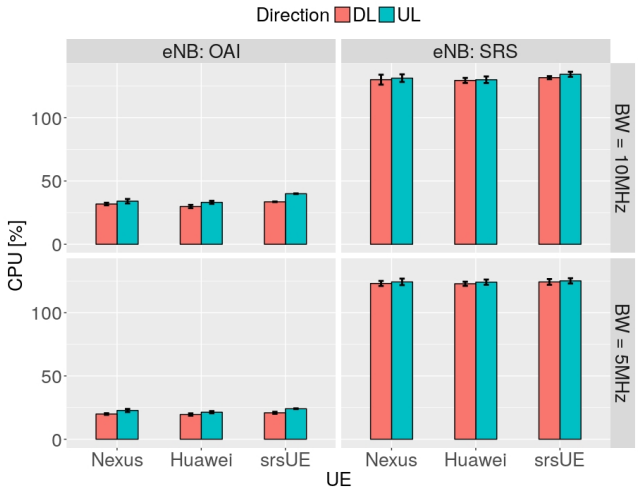

Fig. 4. CPU usage at the eNB for different eNB stacks, UEs, bandwidths, and transmission directions.

The figure confirms that the choice of UE has little impact on resource consumption, as all results are very similar for a given configuration of eNB and bandwidth. It also confirms that there is a notable difference in terms of CPU usage between the SRS and OAI stacks, the former consuming more than four times more CPU cycles than the latter. These results suggest that OAI is more suitable for future 5G scenarios, where computationally efficiency is of paramount importance, such as Cloud RAN. For all configurations, the UL direction is slightly more CPU demanding than the the DL (9.7% on average), which we conjecture is caused by the decoding operations [13].

Finally, it is also worth observing the relative impact of the bandwidth configured on the CPU consumption. Using a simple linear regression model, admittedly built with limited data, we can roughly estimate that OAI may consume on average 2.5% CPU time for every additional MHz of bandwidth, whilst exhibiting an "idling" cost of 9.1% CPU usage. Interestingly,

SRS only appears to add some 1.4% CPU consumption for every additional MHz of bandwidth, but the stack may demand 116.8% CPU time even when the channel bandwidth would be virtually zero.

C. Multiple UEs

We also investigate whether the number of UEs attached to the eNB stacks considered might have an impact on the total throughput performance and CPU usage, due to additional processing that may be required. At the same time, we wish to understand if the schedulers implemented ensure the available resources are shared fairly among the UEs. Therefore, we deploy two UEs that implement the srsUE stack and measure the metrics of interest with both eNBs. The obtained results confirm that adding more clients does not impact on the total network throughput or the CPU usage of neither of the eNB stacks. In addition, both UEs obtain equal throughput, which confirms the accuracy of the schedulers. As a final note, following code inspection we find that OAI implements a proportional fair scheduler, whilst srsENB employs round robin scheduling.

D. Network bootstrap and reliability

Next, we take a closer look at the time required to successfully bootstrap the network setup and how often this process may fail on average. We argue this is particularly important to understand the degree of experiment automation and repeatability achievable with these platforms. To this end, based on the data collected prior to executing the previous experiments, we summarize in Fig. 5 the distributions of the time elapsed until the UE has obtained an IP address and thus has a Layer 3 connection (steps 2–4 described in Section II-B) for the different eNB/UE combinations, over a wireless channel. We resort to box and whisker plots for this purpose, the central lines marking the medians, the boxes' lower and upper margins the 25th and respectively 75th percentiles, and the whiskers the minimum and maximum values, excluding outliers, which we plot separately (crosses).

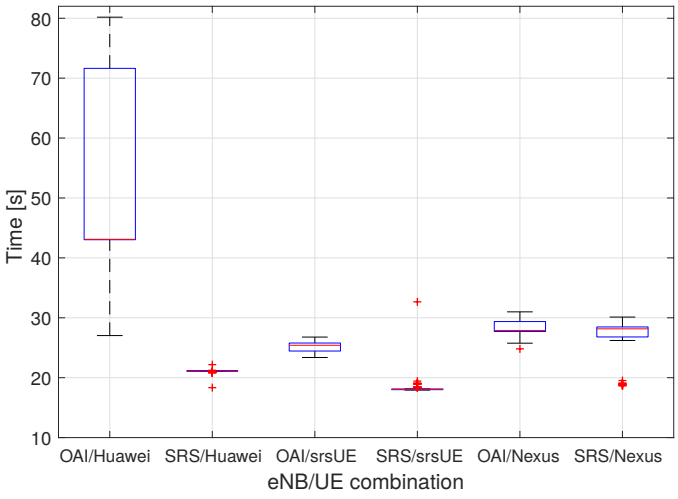

Fig. 5. Distribution of the network bootstrap times (until the UE can ping the CN over the air) with the different eNB/UE combinations considered.

Observe that the SRS stack displays the most deterministic behavior, especially with the Huawei dongle and the SDR running srsUE. Indeed, if we exclude outliers, the bootstrap time is constant and less than 22 s. With the same UE types, the OAI stack performs substantially different. In particular, while the bootstrap time is fairly constant and close in magnitude to that observed when the eNB runs the SRS stack (approximately 25 s median value) and the srsUE is used, the range of bootstrap times is very large when the UE used is the USB dongle (Huawei). In this case, the median is also higher than that measured in all the other setups and the 75th percentile is more than 70 s. This is particularly inconvenient, since when experimenting for performance assessment purposes, setting up the network takes more time than running the actual traffic. In case the Nexus smartphone is used as UE, both eNB types exhibit similar statistics, i.e., the median of the bootstrap time is approximately 28 s and the variations are relatively small (less than 2 s between the 1st and 3rd quartiles). We leave an analysis of the reasons behind these differences for future work.

We also count the number of times we detected a failure of the bootstrap procedure or during experiment runtime for each of the eNB/UE combinations considered, in the process of conducting a total of 80 successful experiments in each case, which we reported earlier (20 repetitions for each direction and bandwidth setting). Interestingly, we find that the most reliable UE type is the srsUE, as we never encountered any failures when connecting this to either of the eNB types used and the network never failed during the experiments. We observed a similar behavior with the USB dongle only when connecting to the OAI eNB stack, while with SRS we measured a 5.9% failure rate. Here, failures occurred always after the network was formed successfully, during the initial short tests that we run to assess the sustainable throughput (step 5 of the set up procedure described earlier). In contrast, in the case of using the Nexus as UE, we observed a 7% failure rate with OAI, as the network did not bootstrap altogether, and a 2.5% failure rate with SRS,, due to network breakdown during experiments.

IV. Extensibility and Pitfalls

A. Customization and extensibility

We next comment on ability to customize and extend the functionality of the platforms considered. To this end, we focus on particular issues related to scheduling and Modulation and Coding Scheme (MCS) assignment. To ensure that the two solutions studied are evaluated under the same conditions and all comparisons are fair, we decided to focus on the following customization: introduce the ability to fix during experiments the MCS assignments which the eNB MAC scheduler enforces on UEs.[13] Achieving this turned out to be fairly intuitive with srsLTE, as we found the function responsible with scheduling and MCS assignment in `srsLTE/srsenb/src/mac/scheduler_ue.cc`, conveniently named `sched_ue` and the code easy to modify.

[13]In the case of srsENB, the software includes code to fix the MCS index at start-up through a configuration file, but our aim is to dynamically change the MCS externally during execution time.

On the other hand, this task turned out to be less straightforward with OAI. The source code that implements the MCS assignment operation is located within the folder `openairinterface5g/openair2/LAYER2/MAC/`, where files related to MAC scheduling and MCS index assignation are located. After thorough code inspection, we found that all the files therein contain code that changes the MCS settings and unfortunately the MCS index is also often hard coded in places. Following debugging, we inferred that the files `eNB_scheduler_ulsch.c` and `eNB_scheduler_dlsch.c` contain the functions `schedule_ulsch_rnti` and `schedule_dlsch_rnti` that assign the MCS in the UL and DL directions. We developed a patch to enable dynamic MCS index assignment, though after applying our patch we discovered that the MCS will be later computed and altered by other functions. Therefore we were unable to achieve the desired behavior.

We believe the major differences between the OAI and srsLTE solutions in terms of extensibility are because they follow different software designs. In particular, OAI was developed for mock LTE network deployment with a built-in emulator, while srsLTE was designed from scratch as a framework to support building LTE applications on top, providing a set of common libraries, tools, and examples for PHY layer implementation and experimentation. As a result, UE and eNB apps were implemented on top of these libraries. From a software design perspective, srsLTE offers a modular framework that re-factors the code of common LTE functions for any application, whilst OAI is designed to offer an standalone eNB solution.

B. Software stack pitfalls

Working with open-source cellular stacks has important benefits, including speed of deployment, availability of documentation, affordable cost, and ability to extend functionality. Unfortunately, such solutions come with their own set of issues, some of which are more difficult to spot and which can hinder the reproducibility of results. Here we highlight the main pitfalls we identified while experimenting with the OAI and SRS tools:

- **Bandwidth incompatibilities**: While SRS supports operation with all the bandwidth settings specified by 3GPP, i.e., 1.4, 3, 5, 10, 15, and 20 MHz, OAI does not work with the 1.4, 3 and 15 MHz configurations. In addition, we find that the srsENB implementation (at least the Sept. 2017 version that we tested) does not work reliably with 20MHz channels. Therefore interoperability between eNBs and UEs running different stacks is limited to only two bandwidth settings, i.e., 5 and 10MHz.
- **Interconnection with CN**: the srsENB implementation employs the same subnetwork for both user plane (S1-U interface) and control plane (S1-C interface). On the other hand, the OpenAir-CN can be configured to use two different subnetworks in order to distinguish between the two planes; if such configuration is enabled, the srsENB stack will not work.
- **Problematic queue management**: We note that sending traffic in the downlink direction, at a rate that exceeds the maximum throughput supported on the channel, makes OAI crash. Following code inspection, we find that the different threads composing the software do not implement any packing dropping strategy at the queues/lists used for communication, which leads to out-of-memory issues. We have fixed this bug and are proposing a patch to the OAI developers community.

V. Summary

This article reports a performance assessment of the two most prevalent open software solutions for mobile network prototyping, namely, srsLTE and OAI. We designed a methodology to characterize the performance of these stacks, quantifying their differences in throughput and resource consumption over a range of practical settings. Our findings formalize "word of mouth" knowledge among practitioners, and provide useful guidelines for deploying 5G testbeds with these tools.

References

[1] 3GPP, "TS 23.501 – System Architecture for the 5G System; Stage 2," Dec 2017.

[2] P. Serrano, P. Salvador, V. Mancuso, and Y. Grunenberger, "Experimenting with commodity 802.11 hardware: Overview and future directions," *IEEE Comms. Surveys Tutorials*, vol. 17, no. 2, pp. 671–699, 2015.

[3] M. Duarte, A. Sabharwal, V. Aggarwal, R. Jana, K. K. Ramakrishnan, C. W. Rice, and N. K. Shankaranarayanan, "Design and characterization of a full-duplex multiantenna system for WiFi networks," *IEEE Trans. Vehicular Tech.*, vol. 63, no. 3, pp. 1160–1177, Mar 2014.

[4] P. D. Sutton, K. E. Nolan, and L. E. Doyle, "Cyclostationary signatures in practical cognitive radio applications," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 1, pp. 13–24, Jan 2008.

[5] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh, "White Space Networking with Wi-Fi Like Connectivity," in *Proc. ACM SIGCOMM*, 2009, pp. 27–38.

[6] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct 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 *Proc. ACM WiNTECH*, 2016.

[8] C. Capretti, F. Gringoli, N. Facchi, and P. Patras, "LTE/Wi-Fi Coexistence under Scrutiny: An Empirical Study," in *Proc. ACM WiNTECH*, Oct 2016.

[9] X. Foukas, M.K. Marina, K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture." In Proc. ACM MobiCom, 2017, 127–140.

[10] X. Foukas, N. Nikaein, M.M. Kassem, M.K. MarinaK. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks." In Proc. ACM CoNEXT, 2016.

[11] A. Virdis, N. Iardella, G. Stea, D. Sabella, "Performance analysis of OpenAirInterface system emulation," PMECT, Rome, Italy, August 2015

[12] N. Nikaein, "Processing Radio Access Network Functions in the Cloud: Critical Issues and Modeling." In Proc. Intl Workshop on Mobile Cloud Computing and Services 2015, 36-43

[13] S. Bhaumik, S. P. Chandrabose, M. K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, and T. Woo, "CloudIQ: A Framework for Processing Base Stations in a Data Center," in Proc. ACM MobiCom, 2012.