Latency Analysis of SDR-based Experimental C-RAN / O-RAN Systems

Christos Laskos, Anatolij Zubow and Falko Dressler
Electrical Engineering and Computer Science, TU Berlin, Germany
{laskos, zubow, dressler}@tkn.tu-berlin.de

***Abstract*—A paradigm shift in radio access network (RAN) architectures was initiated by Cloud RAN (C-RAN) and now Open RAN (O-RAN) systems. Leveraging software defined radios (SDRs), it is possible to enhance scalability, flexibility, and cost efficiency; from prototyping to deployment. However, the feasibility of using SDRs in C-RAN / O-RAN deployments heavily depends on latency constraints, particularly for time-sensitive protocol operations. In this paper, we present results from an extensive experimental evaluation of latency in commonly used SDR platforms, including USRP B210, N210, N310, and X410. For this, we implemented a novel round trip time (RTT) measurement method at the SDR driver level for precise RTT analysis. Our findings highlight the impact of SDR hardware, the connection to the host computer, sampling rates, and protocol optimizations. While all tested SDRs meet the latency requirements for 4G/5G-based C-RAN implementations (less than 500 µs), some approach the 25 µs one-way O-RAN delay, and none currently satisfy the more stringent 9 µs requirements of WiFi. We identify key optimizations, such as reducing the maximum transmission unit used in the fronthaul link layer and leveraging data plane development kit (DPDK) for low-latency networking, that significantly improves the SDR performance. Our results show that, in an optimal setup using X410 USRP, 100Gbe Ethernet fronthaul, DPDK, and optimal maximum transmission unit (MTU) size, the worst-case RTT stays below 65 µs.**

***Index Terms*—Software-defined radio, Cloud-RAN, Open-RAN, Latency, RTT**

I. Introduction

Softwarized radio access network (RAN) technologies promise to enhance cost efficiency, energy savings, network performance, and scalability, enabling rapid prototyping and faster deployments for future networks. Examples include the Cloud RAN (C-RAN) and the Open RAN (O-RAN) architectures. C-RAN is an innovative mobile network architecture that centralizes and virtualizes base station (BS) functions using cloud computing [1]. It consists of distributed Remote Radio Units (RRUs), a high-speed fronthaul network, and a centralized Baseband Unit (BBU) pool. Hence the BS only consists of a RRH, which implements the low-level parts of the physical layer, such as digital to analog conversion and power amplification [1]. A possible input to the RRH is the digital baseband, i.e., I/Q sample stream, from the BBU. An example of a C-RAN architecture is shown in Fig. 1. Similarly, the O-RAN architecture consequently follows the idea of softwarization and allows to distinguish between analog radio frontends and processing using SDR technology [2].

These advantages come with new requirements for the fronthaul network like high-speed, low latency, and low jitter.

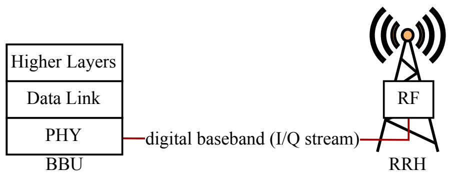

Fig. 1. Cloud-RAN architecture (3GPP split 8 [3])

Failing to meet the timing requirements makes radio transmission and reception inefficient or even impossible. To meet the latency requirements, the Common Public Radio Interface (CPRI) [4] was developed. CPRI typically uses synchronous serial transmission over fiber-optic links and has already been successfully applied to 4G/LTE scenarios [5]. Later, the evolved Common Public Radio Interface (eCPRI) [6] was designed with a packet-based transport in the fronthaul. eCPRI packets are transported inside IP/UDP frames over Ethernet, allowing for more efficient bandwidth usage and flexible deployment. This enables the possibility of implementing C-RAN architectures using conventional software defined radio (SDR) hardware for the RRH combined with Ethernet technology for the fronthaul network. As such, C-RAN testbeds can be developed using SDR hardware for rapid prototyping and evaluation. Similar interfaces are defined for O-RAN.

Depending on the wireless protocol to be evaluated, different latency requirements are to be met. In case of 4G/LTE, the Hybrid Automatic Repeat-Request (HARQ) requires a latency of at most 1 ms in the downlink and 2 ms in the uplink [7]. O-RAN requires between 25 µs and 1 ms one-way latency [8]. In contrast, the IEEE 802.11 WiFi protocol has much tighter latency requirements, where the channel has to be accessed within a 9 µs slot [9]. To examine the feasibility of using SDRs for RAN prototyping, we conducted experiments to investigate latency and jitter of commonly available SDR platforms. We implement a round trip time (RTT) measurement method at the SDR driver level, utilizing the 802.11 waveform for precise RTT analysis, and optimize the default SDR configurations for low latency operation. With our approach, we determine the RTT of an SDR with an actual physical signal transmission.

Our main contribution can be summarized as follows:

- We designed a driver-level method for precise RTT measurements in SDR systems;
- we conducted an extensive measurement campaign for the most widely used USRP SDR systems;

- we identified and optimized latency impacting parameters;
- we derived a latency model that can be used for prototyping SDR-based C-RAN and O-RAN architectures.

II. Related Work

An early SDR latency study was performed by Valentin et al. [10], where a simple send and wait protocol was implemented in GNU Radio for measuring the RTT. The host system sends out a frame through GNU Radio, which is processed and then transmitted via a SDR. The receiving SDR then forwards the received signal to the host, which processes it and sends back another frame. The mean RTT observed was 3.14 ms. Nychis et al. [11] measured the RTT between the kernel on the host system and the field programmable gate array (FPGA) in the SDR using a USB connection. Their effective measured rate is 32 MByte/s, thus, we assume that USB 2.0 was used. Using the default configuration, the mean RTT was 291 µs with a standard deviation of 62 µs. Using an improved configuration with smaller USB transfer block size, they were able to reduce the mean RTT to 148 µs with standard deviation 35 µs. Similarly, Jiao et al. [12] measured the RTT between the host system and the RF frontend in the SDR using the SDR driver latency test, which sends out some stream data to the SDR and measures the time it takes to be acknowledged. They used sampling rates (SRs) of 5, 10 and 25 MHz. Depending on the link technology used in the fronthaul, the latency varies. For USB 3.0 they measured a mean RTT of 66 µs, for PCIe 79 µs, for 1 Gbit/s Ethernet 101 µs, and for 10 Gbit/s Ethernet 106 µs. In an analytical latency study by Molla et al. [13], the RX side latency of an X310 USRP was determined at 4.47 ms, while the minimum link latency is calculated at 2.5 µs.

The wide range of reported latency values from the existing literature motivated us to perform our own measurement campaign to accurately determine both RTT and jitter. Additionally, we aim for further latency reduction by performing a parameter optimization, which allows us to answer the question whether conventional SDRs are suitable for the implementation of C-RAN / O-RAN systems. Going beyond the related work, we implement a precise over-the-air RTT measurement procedure, which includes all sources of latency present in SDR systems.

III. Background

A. C-RAN / O-RAN Architecture

In both the C-RAN and the O-RAN architectures, many functional splits have been proposed by 3GPP [3]. These functional splits determine where the processing steps are performed, e.g., locally at the BS or offloaded to a centralized processing unit [14]. Mainly the baseband unit (BBU) and remote radio head (RRH) are now split into three components. The radio unit (RU) is the RF front end, equivalent to the RRH. The distributed unit (DU) is positioned at the BS and performs tasks locally. The centralized unit (CU) is located at a different site and communicates with the DU. The lowest level split is split 8, where the DU does not have any local processing capacities and the raw I/Q baseband sample stream is forwarded to the CU, where the entire protocol stack is executed.

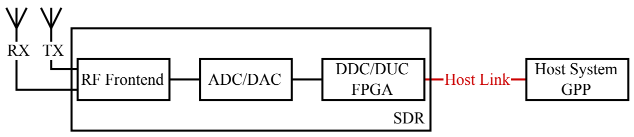

Fig. 2. General software defined radio (SDR) architecture

Next, split 7-1 adds the inverse fast Fourier transformation (IFFT) computation and cyclic prefix addition to the DU, while all other higher level functions are still handled by the CU. More functionality is added to the DU in split 7-2, with beam forming and resource element mapping. In split 7-3, even more functionality is added to the DU, by making it responsible for precoding, layer mapping, and modulation. Split 6 delegates the entire PHY layer processing to the DU and the CU is responsible for the MAC and higher layers. Split 5 adds part of the MAC layer to the DU, such as channel access scheduling. Several more higher level splits exist, which however are not relevant for this work. Split 8 has the highest requirements in terms of data rate, while using split 5 has the lowest. A significant reduction in data rate is already achieved when moving to a 7-1 or 7-2 split. Afterwards, the reduction is not as high as in lower levels.

Considering the WiFi protocol can also be realized using the C-RAN architecture. It requires the split 8 option, but latency requirements are very strict at sub 9 µs for channel access using the carrier-sense multiple access (CSMA) protocol [9]. A more suitable split would be 7-3, where, e.g., IFFT, precoding, and modulation are run locally at the DU. With this approach, many of the time critical tasks would be performed locally. Of course, a split 5, where the entire channel access and PHY layer can be handled locally on the DU, would help further minimizing latency.

B. Software Defined Radio

SDRs are highly programmable devices that are capable of transmitting and receiving radio signals – given certain physical hardware limitations. The programmability is achieved by a connection to a host system, which most commonly uses a general purpose processor (GPP) and communicates to the SDR via Ethernet or USB. The host system performs all the signal processing in software and transmits or receives a stream of samples to the SDR. For example, the entire LTE protocol stack could be run as software on the host system [15]. To support a large signal bandwidth of modern wireless technologies, a high-speed link between the SDR and host is required. This high-speed link connects the host to an FPGA on the SDR, which performs digital down conversion (DDC) and digital up conversion (DUC) to convert in- and outgoing samples to the appropriate SR. Next, analog to digital conversion (ADC) and digital to analog conversion (DAC) take place to transform samples either from or to the analog domain. Finally, the now analog signal is mixed in the RF frontend up/down to/from the set carrier frequency. Figure 2 shows a general SDR architecture, which is equivalent to the 3GPP C-RAN split 8 [3].

Table I
SPECIFICATIONS OF USED SDR CONFIGURATIONS

USRP	SR (MS/s)	fronthaul link technology	required data rate
B210	61.44	USB 3.0 (5 Gbit/s)	$\approx$ 2 Gbit/s
N210	25	1 Gbit/s Ethernet	0.8 Gbit/s
N310	153.6	10 Gbit/s Ethernet	$\approx$ 4.9 Gbit/s
X410	500	100 Gbits/s Ethernet	16 Gbit/s

IV. PRELIMINARY INVESTIGATIONS

In this study, we focus on the universal software radio peripheral (USRP) platform by National Instruments. The selected USRPs and their capabilities are shown in Table I. Note the wide range of sampling rates ranging from 25 to 500 megasamples per second (MS/s). Another distinction between them is the used interconnection to the host system. Mostly Ethernet is used with varying speeds. The B210 is the only USRP using a USB 3.0 connection.

Based on the maximum SR the hardware is capable of, the link technology has to be chosen to support the required data rate. By default, the maximum transmission unit (MTU) between USRP and host is 8000 Bytes, except for the N210, where it is set to 1472 Bytes due to the 1 Gbit/s Ethernet link. With the known maximum SR, we can calculate the minimum required data rate to supply samples to the USRP without the risk of over- and underruns. The default format for transmitting I/Q samples is a 32 bit complex number, consisting of two 16 bit floats for real and imaginary component. With these parameters, we can calculate the required net data rate for reliable sample transmission, without considering protocol overhead and with using only one sample stream (receiving or transmitting): $R_{\text{net}} = S_{\text{R}} \times S_{\text{F}}$, where S_{R} is the sample rate, S_{F} is the sample format (32 bit complex), and R_{net} is the resulting net data rate. The resulting required data rates are shown in Table I. Most of the USRPs have some headroom to perform parameter tweaking on the fronthaul link, i.e., lowering the MTU size. An exception is the N210, which operates near the maximum capacity.

A. Analytical Latency Model

Using an SDR introduces additional latency to the entire transmission and reception process. We analyze the RTT delay from the point in time the radio signal is received at the SDR and processed at the host system, to the point in time the signal is transmitted back by the radio. The RTT can be modeled as follows:

$$\text{RTT} = 2 \times D_{\text{SDR}} + 2 \times D_{\text{COMM}} + D_{\text{PROC}}, \tag{1}$$

where D_{SDR} is the delay introduced by the ADC/DAC and DDC/DUC in the SDR, D_{COMM} is the communication latency of the link between SDR and host, and D_{PROC} is the processing delay on the host system. Here D_{COMM} consists of the following four components:

$$D_{\text{COMM}} = D_{\text{PKT}} + D_{\text{TRAN}} + D_{\text{PROP}} + D_{\text{BUFF}}, \tag{2}$$

where D_{PKT} is the packetization delay, D_{TRAN} is the transmission delay, D_{PROP} is the propagation delay, and D_{BUFF} is the buffering delay. In our study, we attempt to find the model parameters in order to get a deeper understanding of the delays present in state-of-the-art SDR systems.

B. Waveform Generation and Detection

For the RTT measurement, we use an OFDM 802.11a waveform, as it can be used with all USRPs under study. In particular only the short training field (STF) of the PHY preamble is transmitted [9, Section 17.3.3]. For detecting the STF on the receiver, we used the algorithm developed by Chia-Horng [16], which has also been used in the 802.11a GNU Radio implementation by Bloessl et al. [17]. This algorithm calculates the autocorrelation of the incoming sample stream using a double sliding window, where the sample stream is multiplied with a delayed version of itself and then normalized by the power. First, the autocorrelation coefficient a is calculated, where the individual autocorrelation coefficients of the incoming signal s are summed up over a varying window N_{win}:

$$a[n] = \sum_{k=0}^{N_{\text{win}}-1} s[n+k]\overline{s}[n+k+STF_{\text{sym\_len}}], \tag{3}$$

where $STF_{\text{sym\_len}} = 16$ with the default SR of 20 MHz. In [17], N_{win} is chosen to be three times the STF symbol length ($STF_{\text{sym\_len}}$), which in this case is 16, as such $N_{\text{win}} = 3 \times STF_{\text{sym\_len}} = 3 \times 16 = 48$. Next, the correlation coefficient a needs to be normalized by the power of the incoming signal p, where the power p is calculated as follows:

$$p[n] = \sum_{k=0}^{N_{\text{win}}-1} s[n+k]\overline{s}[n+k]. \tag{4}$$

Finally, the normalized correlation coefficient c, ranging from zero to one, is calculated by:

$$c[n] = \frac{|a[n]|}{p[n]}, \tag{5}$$

where $|\ |$ represents the magnitude. For a signal to be detected as the STF, we chose $c[n] \geq 0.75$. As soon as $c[n]$ drops below this level, we consider the frame as over.

V. EXPERIMENTAL SETUP

For our experimental study, we used following hardware:

- **SDR**: USRP B210, N210, N310, X410
- **Host**: AMD Ryzen 9 7950X 16-Core (simultaneous multi threading disabled), 64 GByte RAM, Mellanox 100 Gbit/s MT27800 Family [ConnectX-5], Intel 10 Gbit/s 82599ES, Ubuntu 22.04 LTS, UHD 4.6

Our experimental setup is shown in Fig. 3. Two USRPs (sender and receiver) were connected to one host each. Depending on the USRP, it is either connected through USB 3.0 or Ethernet (1, 10 and 100 Gbit/s). As our study requires both SDRs to transmit and receive simultaneously during measurement, one TX and one RX port on each device is used and is connected to a shared coaxial cable using splitters (S). Using the wired link, outside RF interference is avoided which results in very accurate estimation of the RTT.

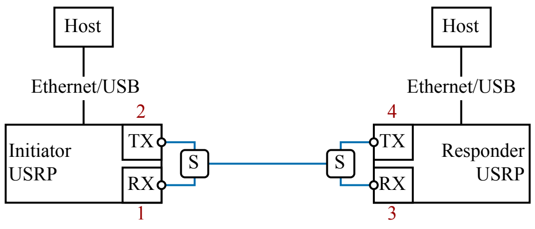

Fig. 3. Setup with USRPs connected via splitters and coaxial cable.

VI. Evaluation

A. Methodology

For a precise RTT measurement, we implemented the STF detection algorithm directly inside USRP hardware driver (UHD) using C++. We measure the RTT by running both SDRs in full-duplex mode, where they can simultaneously transmit and receive. During the measurement phase, the designated responder USRP runs continuously, checking for the reception of the expected waveform. If the waveform is detected, it replies to it as soon as the correlation coefficient c drops below the predefined 75% threshold by sending back the same waveform. The designated initiator USRP initiates the measurement by first starting to receive on its RX port and afterwards sends the precomputed STF on its TX port. The transmitted waveform is then received by the responder on its RX port, processed on the host, and as soon as it ends, the same waveform is send back. When sending back the reply, the responder drops all further received samples to not create an infinite transmission loop. After the predefined receive duration of the initiator (2 ms), the received samples are saved to a file for post processing and the next RTT measurement continues.

With this measurement procedure, we create a ping-pong configuration between initiator and responder to determine the RTT with a per sample accuracy. The measurement procedure is shown in Fig. 4. To actually calculate the time it takes for the responder to reply, we subtract the start time of the responder transmission from the end time of the initiator transmission, in this case $x_1 - x_0$. Using the known sample rate, we convert this into time in seconds. With this procedure, we calculate the RTT, because the signal has to be received at the SDR, transmitted to the host, processed, and then a reply has to be sent back to the SDR from the host.

Due to the homogeneous nature of the different SDRs used in our study, we need to adapt the STF detection algorithm to be able to handle the different SRs. As a first step, the

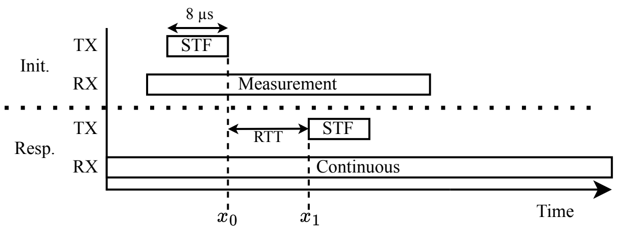

Fig. 4. Procedure for measurement: RTT is measured at the initiator RX port with RTT $= x_1 - x_0$.

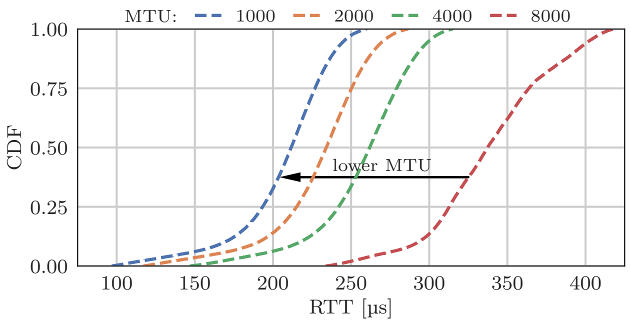

Fig. 5. USRP B210 (USB, SR 20 MHz, variable MTU)

generated waveform is resampled to the correct frequency from the base 20 MHz. We then use SRs that can be adjusted to fit the STF accordingly. This is trivial for SRs of 20, 40 and 500 MHz, where we just down-sample the incoming signal to the base 20 MHz SR by dropping samples. In case of a N310 using a SR of 125 MHz, we change the detection algorithm as follows: First, we down-sample the signal to 31.25 MHz, and then change the $\text{STF}_{\text{sym\_len}}$ in Eq. (3) to 25, to adapt to the new SR. This change is only possible due to $\text{STF}_{\text{sym\_len}} = 16 \times \frac{31.25\,\text{MHz}}{20\,\text{MHz}} = 25$ having an integer result. The equivalent holds for the X410 SR of 250 MHz, down-sampled by ten, with $\text{STF}_{\text{sym\_len}} = 20$.

The parameters evaluated in our study are the following: SR, interface SDR-Host (socket vs. DPDK), MTU size (number of bytes per Ethernet/USB frame), and SDR type (B210, N210, N310, X410). The reported RTT is given as a tuple of minimum (min), median (mdn), maximum (max) RTT values for each configuration.

B. Results

B210: Fig. 5 shows the results for the B210, which is the only device using an USB connection. Using a SR of 20 MHz with the default configuration of a 8000 Bytes MTU, the min, mdn, and max RTT are (234 µs, 337 µs, and 418 µs), respectively. Decreasing the MTU to 1000 Bytes reduces the RTT to (97 µs, 212 µs, and 260 µs). By doubling the SR to 40 MHz, the RTT of the B210 can be improved to (171 µs, 271 µs, and 329 µs). Again, a decrease in MTU results in RTT improvements. For a 1000 Bytes MTU, the worst case RTT slightly increases (95 µs, 196 µs, and 281 µs). A 2000 Bytes MTU shows better results (104 µs, 203 µs, and 257 µs). Due to this increase in worst case RTT and the possibility of dropped samples due to underruns in low MTU configurations, a 2000 Byte MTU is preferred.

N210: Next, we examine the N210, which uses a 1 Gbit/s Ethernet link. Here, the default configuration uses a MTU of 1500 Bytes. We further set SR to 20 MHz. The results are shown in Fig. 6. There is no significant RTT improvement for a reduction in MTU size. All perform similar except the 250 Bytes configuration. In the default configuration, the RTT is (194 µs, 290 µs, and 411 µs), whereas the 250 Bytes configuration results in (150 µs, 284 µs, and 395 µs).

N310: The N310 uses a 10 Gbit/s Ethernet link and supports SRs of up to 125 MHz. Additionally, it supports data plane development kit (DPDK), which replaces the network card

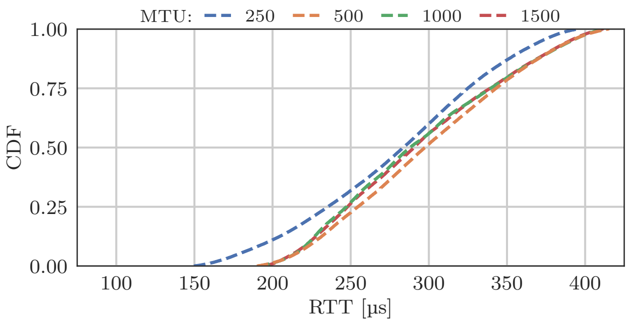

Fig. 6. USRP N210 (1 Gbit/s Ethernet, SR 20 MHz, variable MTU)

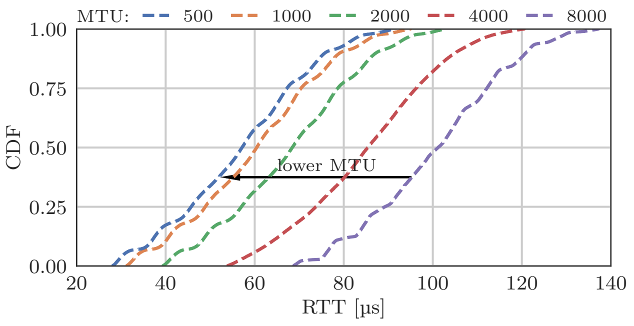

Fig. 8. USRP N310 (10 Gbit/s Ethernet, DPDK,, SR 125 MHz, variable MTU)

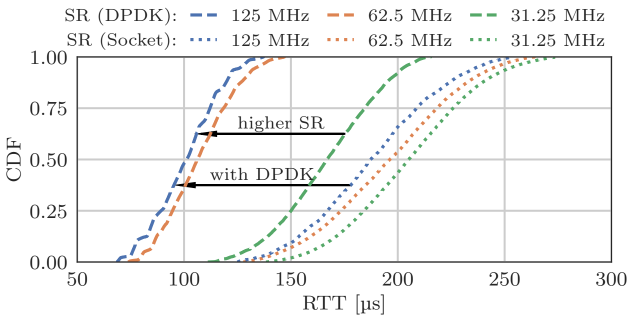

Fig. 7. USRP N310 (10 Gbit/s Ethernet, with/without DPDK, variable SR)

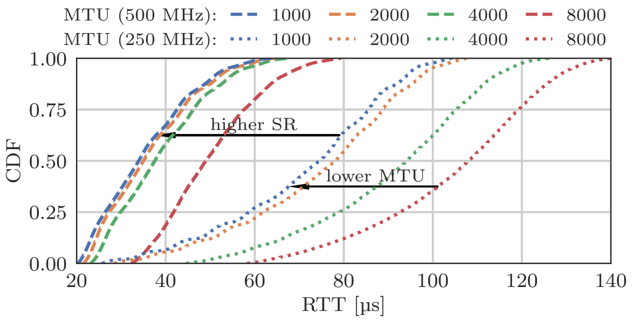

Fig. 9. USRP X410 (100 Gbit/s Ethernet, DPDK, variable SR, variable MTU)

driver and bypasses the operating system network stack. We first examine the impact of using a conventional socket compared to DPDK, as well as the impact of the SR. The results are shown in Fig. 7. We observe a significant reduction in RTT when using DPDK, where the higher SRs benefit the most from it. Comparing 125 MHz DPDK to the socket case, the RTT drops from (125 µs, 187 µs, 254 µs) to (69 µs, 101 µs, 137 µs). DPDK nearly reduces RTT by 50 %. Another interesting observation, is that the RTT does not change significantly when increasing the SR using a socket, while there is a substantial RTT reduction with increasing SR and DPDK. The fairly constant RTT in the socket case, may be caused by the operating system kernel having a high but constant delay. Overall, it is recommend to use DPDK when possible, as it significantly decreases the RTT, especially for higher sampling rates.

We continue with the N310 and examine the impact of the MTU size when using DPDK and a SR of 125 MHz. The results are shown in Fig. 8. Again, we see significant RTT improvements with decreasing MTU, where the default 8000 Byte has an RTT of (69 µs, 101 µs, 137 µs), while the reduced 500 Bytes has an RTT of (28 µs, 57 µs, 91 µs). Just reducing the MTU can reduce the median RTT by over 40 %. We recommend using a 1000 Bytes MTU with DPDK for low latency and avoiding potential sample drops, as the difference to 500 Bytes is only 3 µs.

X410: Finally, we evaluate the X410 and analyze the impact of SR and MTU. We use both maximum available SRs of 250 and 500 MHz, which depends on the FPGA image type. We use DPDK in all cases. The results are shown in Fig. 9. Again, we see similar results as in the previous USRPs, where the RTT is reduced drastically with increased SR and lower MTU. Just increasing the SR with default 8000 Bytes MTU, can reduce RTT from (58 µs, 108 µs, and 140 µs) to (32 µs, 49 µs, and 79 µs). This corresponds to a reduction of roughly 50 %. Lowering the MTU for a SR of 250 MHz shows further improvements until an MTU of 2000 Bytes. The best case RTT for 1000 Bytes MTU is (26 µs, 75 µs, and 104 µs). In the 500 MHz case, only lowering the MTU to 4000 Bytes shows a significant RTT decrease. A lower MTU does not lead to further improvements. Here, the best case RTT is MTU 1000 Bytes) is (20 µs, 34 µs, and 64 µs). In general, it is recommended to adapt the default configuration for the X410 to a lower MTU and increase the SR for latency minimization.

Summary: Fig. 10 shows the results of all USRPs using their best configuration. We observe that the X410 and N310 are better suited for low latency applications. The B210 and N210 on the other hand, have a much higher latency, but may still be useful for application, e.g., LTE, where higher RTT can be tolerated.

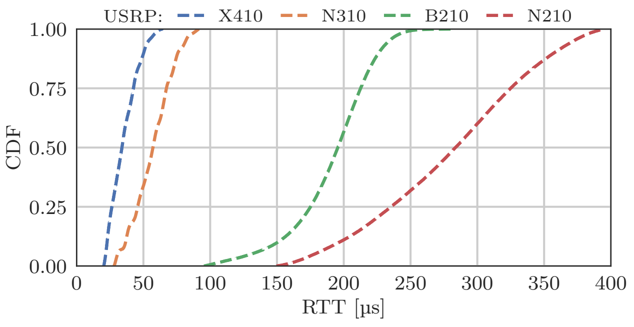

Fig. 10. Comparison of different USRPs using the best configuration

VII. Latency Analysis

After our extensive measurement campaign, we now aim to solve Eq. (1) for a specific configuration. We consider the N310 USRP with 125 MHz SR, 10 Gbit/s Ethernet link, and usage of DPDK. We assume that the measured RTT is symmetric. In a first step, we subtract the processing delay from the RTT measurements, which we measured at $D_{\text{PROC}} \leq 1.7\,\mu\text{s}$. We divide the RTT by two to get the one-way delay and subtract the SDR delay D_{SDR}, which we assume to be equal to the measurement results reported by Bräuer et al. [18], i.e., 5.1 µs, when using DDC/DUC. The last unknown is the communication delay D_{COMM}. We have to analyze each of its components individually. We first calculate the packetization delay D_{PKT} based on the SR and the sample size $s_{\text{size}} = 32\,\text{bit}$ with $D_{\text{PKT}} = \frac{\text{MTU}}{\text{SR} \times s_{\text{size}}}$. For MTUs of 8000, 4000, 2000, 1000, and 500 Bytes we get an approx. $D_{\text{PKT}} = 16,\ 8,\ 4,\ 2$ and $1\,\mu\text{s}$ (rounded to µs). Next, we analyze the transmission delay D_{TRAN}, with $D_{\text{TRAN}} = \frac{P_{\text{size}}}{R}$, where P_{size} is the packet size including all headers (UHD, UDP, IP, Ethernet) and R is the data rate of the link with $R = 10\,\text{Gbit/s}$. For our different MTUs, we get an approx. $D_{\text{TRAN}} = 6,\ 3,\ 2,\ 1$ and $0.5\,\mu\text{s}$. D_{PROP} is calculated with $D_{\text{PROP}} = \frac{l}{(2/3) \times c} = 10\,\text{ns}$, where $l = 2\,\text{m}$ and c is the speed of light. We attribute the rest of our occurring delay to the remaining term D_{BUFF}. As the buffering delay is not a constant like all other delays, it follows a random distribution. We perform a distribution fit and determine that D_{BUFF} follows a normal distribution, which can be parameterized as follows:

$$D_{\text{BUFF}} = \mathcal{N}(23, 8)\,\mu\text{s} \tag{6}$$

This completes the latency model for the N310 USRP so that it can be used in simulation studies.

VIII. Discussion and Conclusions

We investigated the latency and jitter introduced by the widely used SDR platforms by means of extensive experiments. We found that the RTT depends mainly on the SDR used. Moreover, the used configuration like the SR has a large impact. We discovered that for all platforms under study significant latency and jitter reductions can be achieved by tuning the MTU size of the fronthaul link, i.e., Ethernet or USB, as well as increasing the used SRs. We found that the biggest contributor to latency is the communication link between SDR and host whereas the latency inside the SDR, i.e., up/down conversion, and host, i.e., processing delay, play only a minor role. Moreover, the use of DPDK reduced the RTT by 50 %. We now have a good understanding of the different sources of latency in a SDR system. All examined SDRs have the potential to be used for LTE C-RAN prototyping, some for O-RAN prototyping, but unfortunately none are able to satisfy the very low latency requirements of the WiFi protocol.

Acknowledgments

This work was supported by the Federal Ministry of Education and Research (BMBF, Germany) within the 6G Research and Innovation Cluster 6G-RIC under Grant 16KISK020K.

References

[1] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for Mobile Networks—A Technology Overview," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 405–426, Jan. 2015.

[2] M. Polese, M. Dohler, F. Dressler, M. Erol-Kantarci, R. Jana, R. Knopp, and T. Melodia, "Empowering the 6G Cellular Architecture with Open RAN," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 245–262, Feb. 2024.

[3] "Study on new radio access technology: Radio access architecture and interfaces," 3GPP, Sophia Antipolis, France, TR 38.801 V14.0.0, Mar. 2017.

[4] "Common Public Radio Interface (CPRI)," Ericsson AB, Huawei Technologies Co. Ltd, NEC Corporation, Alcatel Lucent, and Nokia Networks, Interface Specification V7.0, Oct. 2015.

[5] A. de la Oliva, J. A. Hernandez, D. Larrabeiti, and A. Azcorra, "An overview of the CPRI specification and its application to C-RAN-based LTE scenarios," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 152–159, Feb. 2016.

[6] "Common Public Radio Interface: eCPRI Interface Specification," Ericsson AB, Huawei Technologies Co. Ltd, NEC Corporation and Nokia, Interface Specification V2.0, May 2019.

[7] V. Q. Rodriguez, F. Guillemin, A. Ferrieux, and L. Thomas, "Cloud-RAN functional split for an efficient fronthaul network," in *International Wireless Communications and Mobile Computing (IWCMC 2020)*, Limassol, Cyprus: IEEE, Jun. 2020.

[8] E. Municio, G. Garcia-Aviles, A. Garcia-Saavedra, and X. Costa-Pérez, "O-RAN: Analysis of Latency-Critical Interfaces and Overview of Time Sensitive Networking Solutions," *IEEE Communications Standards Magazine*, vol. 7, no. 3, pp. 82–89, Sep. 2023.

[9] IEEE, "Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: High Speed Physical Layer in the 5 GHz band," IEEE, Std 802.11a-1999, Dec. 1999.

[10] S. Valentin, H. von Malm, and H. Karl, "Evaluating the GNU Software Radio platform for wireless testbeds," Paderborn University, Paderborn, Germany, Technical Report TR-RI-06-273, Feb. 2006.

[11] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC Protocol Implementations on Software-Defined Radios," in *6th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2009)*, Boston, MA: USENIX, Apr. 2009, pp. 91–105.

[12] X. Jiao, I. Moerman, W. Liu, and F. A. P. de Figueiredo, "Radio Hardware Virtualization for Coping with Dynamic Heterogeneous Wireless Environments," in *12th International Conference on Cognitive Radio Oriented Wireless Networks (CrownCom 2017)*, Lisbon, Portugal: Springer, Sep. 2017, pp. 287–297.

[13] D. M. Molla, H. Badis, L. George, and M. Berbineau, "Software Defined Radio Platforms for Wireless Technologies," *IEEE Access*, vol. 10, pp. 26 203–26 229, Feb. 2022.

[14] L. M. P. Larsen, A. Checko, and H. L. Christiansen, "A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2019.

[15] 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 *22nd ACM International Conference on Mobile Computing and Networking (MobiCom 2016), 10th ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization (WiNTECH 2016)*, New York City, NY: ACM, Oct. 2016, pp. 25–32.

[16] L. Chia-Horng, "On the design of OFDM signal detection algorithms for hardware implementation," in *IEEE Global Telecommunications Conference (GLOBECOM 2003)*, San Francisco, CA: IEEE, Dec. 2003, pp. 596–599.

[17] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, "An IEEE 802.11a/g/p OFDM Receiver for GNU Radio," in *ACM SIGCOMM 2013, 2nd ACM Software Radio Implementation Forum (SRIF 2013)*, Hong Kong, China: ACM, Aug. 2013, pp. 9–16.

[18] S. Bräuer, A. Zubow, and F. Dressler, "Towards Software-Centric Listen-Before-Talk on Software-Defined Radios," in *IEEE Wireless Communications and Networking Conference (WCNC 2021)*, Nanjing, China: IEEE, Mar. 2021.