Latency Analysis in GNU Radio/USRP-based Software Radio Platforms

Nguyen B.Truong
DASAN NETWORKS Corp.
Seoul, Gyeonggi-do, 463-400
South Korea
nguyentb@postech.ac.kr

Young-Joo Suh
Department of Computer Science and Engineering, POSTECH
Pohang, Gyeoungbuk-do, 790-785
South Korea
yjsuh@postech.ac.kr

Chansu Yu
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, OH 44115, U.S.A
c.yu91@csuohio.edu

***Abstract*— Software-Defined Radio (SDR) is a promising radio technology that implements radio communication functionalities in software instead of hardware. Advantages of a SDR system are reconfigurability and flexibility whereas disadvantages are low throughput and high latency. Our research focuses on analyzing and measuring latency of a SDR system based on GNU Radio with three versions of Universal Software Radio Peripherals (USRP) devices – USRP1, USRP2 and USRP E100. This research identifies the sources of the latency and quantifies them by using both analytical and experimental methods. For the analysis, we identify all types of buffers in the SDR platforms and estimate the time that these buffers contribute. The corresponding results are compared with experimental measurement, which estimates the round-trip time between two SDR systems. This can be done by using TUN/TAP components and GNU Radio *tunnel* program. Alternatively, we also propose a method called *hwlatency* to measure the latency in hardware side. We believe this study offers the better understanding of the latency in SDR platforms and will lead to correct implementations of high-level network protocols for SDR systems.**

***Index Terms*—Software Defined Radio, SDR, GNU Radio, USRP, latency, buffer, TUN/TAP.**

I. Introduction

Conventional radio communication systems usually contain antennas and components for modulation, de-modulation, detectors, amplifiers, filters and mixers which are implemented in hardware. A software-defined radio (SDR) is a radio communication technology that, instead, implements most of those components in software, except the antennas. Thus, it is advantageous in terms of reconfigurability and flexibility and is considered an attractive solution for military and disaster response applications. In these scenarios, radio communication channels are either intentionally or unintentionally destroyed or damaged but the SDR system can adjust itself to survive and maintain the necessary communications. The corresponding capability is also known as Cognitive Radio (CR).

To satisfy real-time processing and timing requirements of most of state-of-the-art wireless protocols, some SDR platforms implement digital signal processing (DSP) functions on field programmable gate arrays (FPGAs) and embedded digital signal processor (DSPs) [4, 5]. However, they require longer development time and higher level of efforts and are expensive. For example, Wireless Open Access Research Platform (WARP) costs more than US$9,750 [5].

On the other hand, SDR platform architectures based on general-purpose processor (GPP) or a combination of GPP and DSP are relatively inexpensive, i.e., an Universal Software Radio Peripheral 1 (USRP1) costs about US$1,000 to US$1,500 [1]. Moreover, they are advantageous in terms of programming environments and tools. However, the use of this type of SDR systems in real world is very limited due to its limited performance. These architectures comprise of multiple heterogeneous processing units with interconnecting buses leading to a large amount of delay and a low throughput. For example, USRP1 with GNU Radio running on top of Intel Pentium IV desktop PC, achieves the maximum throughput of 8Msps (samples per second). This is because USRP1 is connected to the PC through USB, which serves as a bottleneck [8, 9, 10]. With 16-bit complex (16-bit I and 16-bit Q sample, 4 bytes) samples, this SDR system can cover only 7.5 MHz channel because the bandlimit is no greater than one half of the sampling rate and the maximum USB rate is 60MB/sec [10]. The maximum throughput is at 120Kbps [8] that is far from the current high-speed wireless protocols, such as IEEE 802.11 requiring several Mbps data rate on 20 MHz channel [6].

Thus, SDR based on GPP architectures needs to deal with delay-related challenges. First, transmitting digital samples from the hardware device to a host computer memory requires high bus communication throughput. Second, some DSP tasks require high computational complexity due to a huge amount of data to be extracted, processed or generated from high-fidelity waveforms at high modulation rates. Third, physical layer and MAC layer protocols require nearly real-time processing with very strict deadlines on the orders of several micro-seconds. Currently, most of GPP-based SDR platforms do not satisfy these strict timing requirements [8, 9, 10].

This article examines the delay on one of the most popular SDR platform, GNU Radio/USRP, to find out the sources and the quantity of the latency, thus, to help decrease the delay in future SDR systems. Section II explains GNU Radio and USRP systems, which is followed by latency analysis in Section III.

978-0-7695-5124-1/13 $31.00 © 2013 IEEE
DOI 10.1109/MILCOM.2013.60

Experimental testbed and results are presented in Section IV and V, respectively. This paper concludes in Section VI.

II. Background and Related Work

A. GNU Radio software

GNU Radio is a free, open-source toolkit containing libraries to conduct DSP functions and utilities. It utilizes several RF front-end daughterboards with a FPGA-based main system to implement a variety of real world radio systems. GNU Radio can also work alone without real RF devices as a simulation environment by using pre-recorded or generated data to create a simulation environment [2].

GNU Radio applications are commonly written in Python as a wrapper and combine DSP blocks integrated in GNU Radio. The DSP blocks implemented in C++ perform critical-performance DSP tasks such as modulations, demodulations, filters, mixers, and signal operators. A generator named Simplified Wrapper and Interface Generator (SWIG) is used to interface between Python programs and the DSP blocks [2]. This architecture promotes the reuse of existing DSP blocks and makes programming easy by using Python script.

B. Universal Software Radio Peripheral (USRP)

USRP is an inexpensive and flexible hardware platform for software radio from Ettus Research. USRPs are widely used in research labs, universities, and industry. An USRP consists of analog digital converters (ADCs), digital analog converters (DACs), RF antenna and a FPGA which can be programmed to implement some DSP functions and digital down converter (DDC) block. USRPs also contain bus controllers for transferring samples from/to a host computer via USB 2.0 (USRP1) or Gigabit Ethernet (USRP2). New USRP Embedded series integrates general functionality of a host computer within the USRP to allow it to operate in a standalone fashion. USRP Embedded series has General Purpose Memory Controller (GPMC) bus controller for communicating between embedded processor and FPGA [13]. Along with USRPs are daughter-boards that are RF front-ends for sending and capturing radio signals with the ability of tuning to different frequency bands.

C. Universal Hardware Driver (UHD)

UHD is a hardware driver library for all USRP series and all types of daughter-boards. Conventionally, radio software uses different libraries for different USRP series and daughter-board. Instead, UHD provides the consistent Application Program Interface (API). It is possible to use UHD driver standalone or with other applications such as GNU Radio, Labview and Simulink [16]. UHD finds devices on a USRP system and instantiates a device object with desired parameters. It sets /gets radio properties (e.g. gain, amplitude, center frequency, sample rate, and time) and transmits samples by using standard Operating System (OS) *read*() and *write*() operations. To send and receive samples from/to the USRP, UHD creates a sending or receiving stream between the host computer and the FPGA in the USRP. UHD also supports control and management messages such as Overflow, Stream command error (Rx path) such as Underflow, and Sequence error (Tx path). UHD functionalities are wrapped into GNU Radio by using USRP source and sink blocks with *Multi-Usrp* module as shown in Fig. 1.

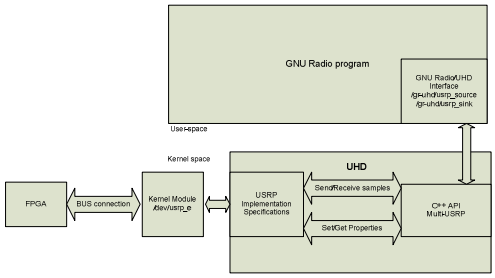

Fig. 1. UHD in GNU Radio/USRP SDR Platforms

D. Related Work

Several previous works measured the latency on the GNU Radio/USRP platforms [7]. However, they concentrated on the latency introduced at the communication bus and have not analyzed sources of the delay in an extensive manner. Valentin et al. measured the round-trip time (RTT) on a testbed consisting of two GNU Radio/USRP1's [9]. It is between the moment the first byte passes the first sender signal block and the moment the first byte (ACK) arrived at the last receiver signal block (head-to-head latency). And it is 3.14ms on the average. While this study does not explain the sources of the delay, it explains that the contribution due to the USB bus transfer and USRP operations is ignorable.

Another experimental study by Schmid et al. showed that this is not the case [8]. An external square wave generator is used, where one output is connected to an oscilloscope and the other connects to a GNU Radio/USRP platform. They measured the difference between two channels of the oscilloscope as the latency of the SDR system. They took Rx latency and Tx latency separately, and the Rx latency is considered as (USRP latency + USB latency + GNU Radio latency). The results showed that the Rx latency is dominated by the USB latency and ranges from 1~30ms depending on the sampling rate used. The Tx latency is measured as 28.9~36.9ms. They identified buffering schemes at GNU Radio and USB driver as important contributors to the latency.

Nychis et al. had a different perspective of latency by looking from the system view [10]. The latency is explained as the sum of the following operations, (GNU Radio processing + GNU Radio to Kernel + Kernel to FPGA + FPGA to Kernel + Kernel to GNU Radio + GNU Radio processing). They proposed a timestamp method, where a low-level ping command was used and timestamp is put at various places in the Tx and Rx path. They showed that the total latency of the SDR system is dominated by the Kernel-to-FPGA-to-Kernel and the GNU Radio processing at the sender and the receiver (291 and 270us, respectively). However, while both GNU Radio-to-Kernel and Kernel-to-GNU Radio have a modest average delay (24 and 27us, respectively), the latter has a very high standard deviation (13~7,000us) [10], possibly due to the OS scheduling mechanism.

In the meantime, researchers realized that GNU Radio/USRP platforms are not efficient enough to implement modern wireless protocols due to high latency and poor performance. Alternative SDR platforms such as SODA [11], SORA [7], and CalRadio [12] have been introduced.

III. Latency Analysis on GNU Radio/USRP Platforms

In this section, we identify the sources of the latency introduced in GNU Radio/USRP SDR platforms and estimate the amount of time that these sources contribute.

A. Main Causes of Latency on GNU Radio/USRP Platforms

Latency on GNU Radio/USRP platforms can be divided into three components: (i) latency introduced in GNU Radio and OS kernel, (ii) latency at communication bus between host computer and USRP, and (iii) latency at USRP hardware.

The first is due to process DSP functions such as producing, filtering, encoding, and modulating data into samples (Tx process); demodulating data, decoding (Rx process); and queuing at GNU Radio buffers. It also takes time for samples to be transmitted between GNU Radio (user-space) and bus driver (kernel-space). This is because GNU Radio might not commit samples to the bus driver buffer immediately; instead, it might need to wait for other higher priority tasks to be processed. The second is introduced because samples are not immediately transferred to the bus; instead, they are put in buffers in order to collect enough data to fill the buffer and to generate packets (USB, Ethernet or GPMC packets) before transmitting. It also takes time to transmit data through the communication bus. The third is that USRP gets samples from the bus, puts them to the internal bus controller buffer, and sends them to the FPGA to be interpolated before sending out via antennas (Tx process). The Rx process is similar.

B. Latency in GNU Radio and OS Kernel (ΔGNU Radio)

- OS dedicates not only to the radio software and bus communication, but also to other processes with different priorities, resulting in an unpredictable waiting time for interacting with the kernel. There is a GNU Radio real-time scheduling mechanism in which the priority is set to be highest, thus, reduces the waiting time.
- Another contributor of ΔGNU Radio is GNU Radio FIFO buffer (default size is equal to the OS page size, which is 32kB and 64KB on Linux and Windows, respectively). Since GNU Radio has the flow-graph design and contains several blocks connecting in serial, each block processes data in previous buffer and generates data to the next buffer as shown in Fig. 2. Thus, if the buffer size increases, the throughput increases but the latency also increases; and vice versa. The process to generate data to fill GNU Radio buffer is not rate limited. Processor tries to fill the buffer as much as it can. For example, to transmit 32kB data with default GNU Radio 32kB buffer, assume that host computer generates 16-bit complex samples at 4Msps. Thus, it takes 2ms to fill the buffer before transmitting to UHD. It also takes time to move these samples between user-space and kernel-space, from 13us to 7ms [10]. By using real-time scheduling mechanism in GNU Radio, this time could be made small and negligible.
- DSP tasks in GPPs may cost a large amount of time, depending on the complexity of the signal processing tasks and power of GPPs. Note that the corresponding delay should not be counted if we're interested in the head-to-head latency mentioned earlier [7]. However, it should be considered if we're interested in the latency until the completion of the data reception.

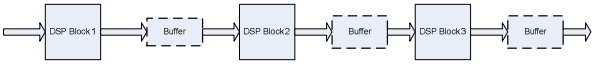

Fig. 2. Buffers on GNU Radio Flow-graph structure

C. Latency at Communication Bus

This latency consists of latency at UHD buffer (ΔUHD), at USRP buffer (ΔUSRP_Buffer) and transfer time through the communication bus (ΔBus),

- UHD buffer holds samples to generate packets before sending to the bus (Tx path) or unpack packets before sending to GNU Radio (Rx path). Thus, ΔUHD is the time to collect enough samples to fulfill the UHD buffer (Tx path) or the time to unpack packets (Rx path). UHD uses libusb1.0 for USB 2.0; Berkeley socket for Ethernet; and a designed GPMC protocol for memory-FPGA communication in USRP E100. Default packet sizes for these bus protocols are 512B, 1,500B and 2,048B, respectively. Default UHD buffer size is 4kB in USRP1 and USRP E100 and is 8kB in USRP2. The buffer size is recommended as the multiple of packet size and can be set through *send_buff_size* and *recv_buff_size* parameters.
- An UHD utility called *benchmark_rate* is conducted to measure ΔBus. The maximum rates for USB2.0, Gigabit Ethernet and GPMC buses are 30Mbps (8Msps), 100Mbps (25Msps), and 48Mbps (12Msps), respectively. For instance, to transmit 2kB data through the three bus systems, ΔBus is 0.512ms (2kB×8/32Mbps), 0.16ms (2kB×8/100Mbps), and 0.333ms (2kB×8/48Mbps), respectively. Note that ΔBus is not a part of the head-to-head latency.
- ΔUSRP_Buffer is due to the 2kB Tx/Rx FIFO buffers in USRP devices. In USRP1, although the buffer size is 2kB, the USB FX2 controller implements quad-buffering in both TX and RX directions. Each sub-buffer is 512B which is equal to USB packet size. Whenever one 512B sub-buffer is full, data is transmitted without the need for fulfilling the 2kB buffer. USRP E100 uses memory-mapped ring mechanism for the communication between USRP FPGA and CPU memory over the GPMC bus. The size of each element in the ring buffer is 4kB but each element consists of two 2kB blocks. Whenever a block is filled, it is activated to transmit by a polling mechanism without waiting for the whole 4kB buffer to be full.

D. Latency in USRP device

In Tx path, when USRP buffer associated with the bus receives a packet, the bus controller immediately transfers the

packet to USRP FPGA buffer. It is then interpolated at FPGA Codecs, digitized at DAC, and sent out via antenna. However, an additional delay is incurred at the 4kB Tx/Rx FPGA FIFO buffers, denoted as Tx/RxΔFPGA. TxΔFPGA is very small and negligible since the processes are implemented in hardware with high-speed system bus between USRP buffers and FPGA buffers. In Rx path, on the other hand, samples are generated then put into FPGA buffer for producing packets before sending to USRP buffer. Thus, RxΔFPGA is proportional to USRP sample rate assuming that the processing time at FPGA Codecs and ADC and system bus delay are negligible.

E. Summary

Figure 3 shows the sources of latency of the Tx path of USRP/GNU Radio platforms. In this figure, the *buffer element* is a *packet-size* amount of the whole buffer. ΔUSRP_Buffer and TxΔFPGA are not shown because they are relatively small and negligible in the Tx path.

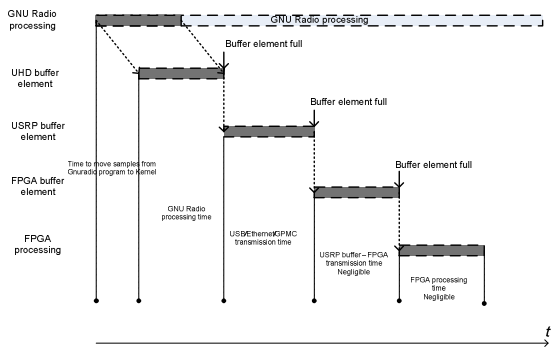

Fig. 3. Sources of Tx latency on GNU Radio/USRP platforms

Figure 4 shows the latency analysis on Tx 512B data on USRP1 and 2kB on USRP1 and USRP E100 using continuous streaming mode assuming that the processor constantly generate samples at 2Msps.

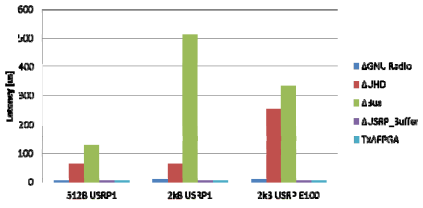

Fig. 4. Transmit Latency Analysis on GNU Radio/USRP platforms with different amount of data

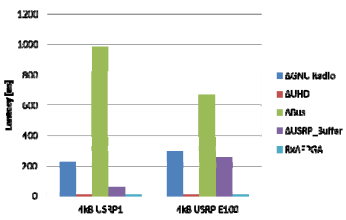

Fig. 5. Receive Latency Analysis on GNU Radio/USRP platforms

Figure 5 is the analysis on Rx 4kB data on USRP1 and USRP E100 platforms at 2Msps USRP sample rate. The analysis indicates that packet size plays an important role in the Tx ΔUHD Buffer or ΔUSRP_Buffer. As packet size decreases, the latencies decrease. The ΔBus dominates the latency in the two cases that we conducted. Thus, the high-speed bus is necessary for reducing the latency.

IV. Experimental Latency Measurement Methods

Two methods are proposed to measure the latency in more detail. The first method is to measure the RTT between two SDR platforms like in [7]. The second method called *hwlatency* measures the delay between host computer and FPGA in USRP device similarly approached in [10].

In the first method, a utility called *tunnel* in GNU Radio which allows tunneling any kind of IP traffic through the SDR system is used. Upon startup, the application creates a virtual Ethernet interface (i.e., gr0) by opening the TUN/TAP modules which is integrated in Linux kernel. Fig. 6 illustrates how the Linux *ping* works in GNU Radio/USRP SDR system.

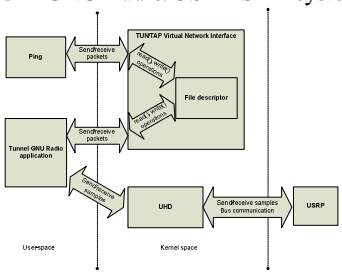

Fig. 6. Linux "ping" utility with GNU Radio tunnel application using TUN/TAP components

Ping generates ICMP request packets and sends to TUN/TAP interface (gr0). The tunnel takes over the packet and sends to the network stack by injecting the packet using OS *"write()"* operation. In Rx path, once USRP receives a packet, it revokes by a thread associated with PHY and passes the received packet up to the TUN/TAP interface via "*read()"* operation. Thus, RTT includes application latency (Linux processes, network stack, TUN/TAP components, read/write file descriptor) and SDR platform latency.

The second method, *hwlatency,* measures the transmission time between host computer and FPGA which consists of bus communication latency and USRP device latency called hardware latency (ΔHardware). The idea is to transmit some amount of data in Rx FPGA buffer to UHD buffer, then estimate the time it takes for the successful transmission. An *UHD received stream* is issued to get samples from the USRP FPGA buffer and send them to the UHD buffer at host computer. After some time *rt,* we issue an *UHD sending stream* to send samples which are already at the UHD buffer. By comparing the number of received samples from FPGA buffer and the number of sent samples, we can deduce that after *rt*, the transmission from FPGA buffer to the UHD buffer is successful or not as depicted in Fig. 7. To get the approximate *rt*, a trail and fail method is used with upper and lower values (*rt1*, *rt2* in Fig.7).

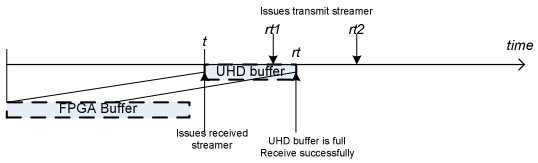

Fig. 7. Idea of the *hwlatency* method

With the assumption that all samples are already generated and put into the FPGA buffer (thus, ΔFPGA = 0), the time *rt* is the sum of ΔUHD, ΔUSRP Buffer and ΔBus. A small trick in the *hwlatency* program is that the *UHD received stream* is not immediately issued. Instead, it needs to be waited for a while in order for USRP to have enough time to generate samples and put them into the FPGA buffer. Since UHD supports management and control messages, we just need to see the ACK in the form of *async_metadata* message to get the results.

V. Evaluation and Measurement Results

For the experiment on USRP1 and USRP2, a host computer is equipped with Intel core-i5 clocked at 2.93 GHz, 4GB RAM, USB 2.0, and Gigabit Ethernet, running Ubuntu default kernel 10.04 LTS. USRP E100 is equipped with 720 MHz OMAP3 (Arm Cortex A8 processor & TI C64x+ DSP) embedded microprocessor; 512Mb RAM; running a simplified Linux (E1xx-003 image, provided by Ettus Research). This image is integrated with GNU Radio ver. 4.5.3 and UHD ver. 3.4 (UHD_003.004.000). All three USRPs are equipped with RFX2400 daughterboard, operating in 2.4GHz ISM band.

The distance between USRP receiver and USRP transmitter is about 2m. Frequency calibration is conducted to avoid the frequency offset, which frequently occurs during experiment on real hardware devices. Amplitude/Gain adjustment was also applied to improve the successful rate of the transmission. To measure the RTT of USRP1/USRP2 GNU Radio platforms, we sent 400 ICMP packets using standard Linux *ping*. The successful rates were more than 95%. We randomly chose 300 successful transmissions of ICMP packets to calculate the statistics. For USRP E100, we sent 1,000 ICMP packets and got about 170 successful transmissions. The successful transmission rate (17%) was small because of the CRC check for the received packets almost failed. This might be because of limited processing power of the Arm Cortex A8 coprocessor. Moreover current GNU Compiler Collection does not well optimize for NEON SIMD in the coprocessor resulting in poor performance on the system. We randomly chose 100 out of 170 successful transmissions to calculate the statistics. The parameter UHD buffer size was put into account by setting up the parameter *"sent_buf_size"*, and *"recv_buf_size"* when initiating a new instance of USRP. We choose the buffer size among five values: 1,024, 2,048, 4,096 and 16,384 byte. The ICMP packet header at the MAC layer is 64B and we added 1,984B payload more. The packet size is 2kB in total.

The effect of changing UHD buffer sizes on GNU Radio/USRP1 platform was mentioned in [18, 19]. Our experiment shows that Tx UHD Buffer *"sent_buf_size"* does not play any role in the Tx latency. It is straightforward because once samples are collected enough to generate a packet, it is sent right away without the need for Tx UHD buffer getting full. On the other hand, "*recv_buf_size"* plays an important role to the Rx latency because samples received from the bus are put into this buffer. Once Rx UHD buffer is filled, samples are transmitted to the GNU Radio program.

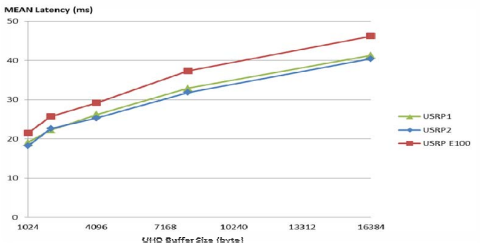

Fig. 8. Mean Latency comparison among three USRP platforms

The average RTTs of the USRP1 and USRP2 SDR platforms are very similar. Overall, the GNU Radio/USRP2 platform is little faster but not much, about 1ms. The RTT of the USRP E100 platform is relatively higher than the rest for most of the cases (Fig. 8).

For measuring ΔHardware, we conducted 1,000 tests with different *rt* time values for three types of USRPs. For each *rt*, we calculated both the number of successful transmissions and the number of failures. To compare analytical latency results with the experimental RTT, we choose the same parameters: 512 samples (2kB) to transmit at 2Msps sample rate. In case of USRP2 and USRP E100, the experimental hardware latency is similar to analytical results. However, the experimental latency of USRP1 is about 6 times higher than the analytical result (Fig. 9). It may be that the data rate on USB2.0 is smaller than the maximum 32Mbps that we assume. Another possibility is that the processing at FPGA in USRP1 is non-negligible.

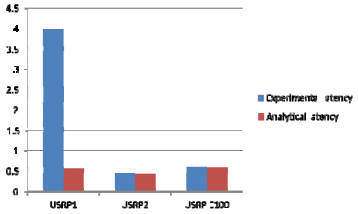

Fig. 9. Experimental and Analytical Hardware Latency

We analyzed the relationship between RTT and hardware latency by assuming Tx latency is same as Rx latency when "recv_buf_size" is 2kB. In this case, whenever an ICMP packet is received, the Rx UHD buffer gets full and immediately transfer the data to GNU Radio. The packet time over the air is also considered as negligible, thus, Tx latency = Rx latency = RTT/4 because RTT = 2*(Tx + Rx). Tx latency could be divided into two parts: Application latency and SDR platform latency. Application latency is the delay introduced in the application but not related to the SDR platform such as in "*ping*" command or in TUN/TAP creation. SDR platform latency consists of ΔGNU Radio and ΔHardware (Fig. 10). The ΔGNU Radio in this case is roughly considered as the time it

takes to read TUN/TAP file descriptor, process and generate 2kB ICMP packets into samples.

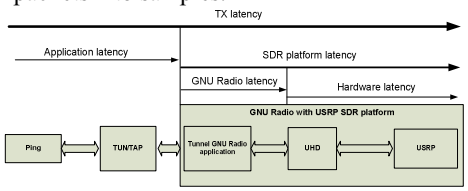

Fig. 10. RTT analyze in details

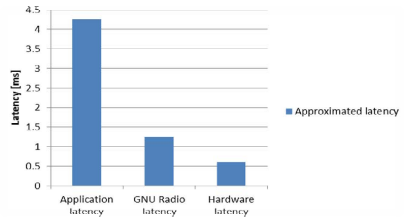

Fig. 11. Approximated Latencies for TX path

Assume that the GNU Radio can process samples at the rate 2Msps in USRP E100, thus, ΔGNU Radio is at least 0.256ms in order to generate a 2kB GPMC packet. The Tx latency is equal to RTT/4 = 6.125ms. The ΔHardware is about 0.61ms, the GNU Radio latency is at least 0.256ms. It also takes time for reading file descriptor, which consumes a lot of time due to disk accessing and user-space/kernel-space switching. Assume that this task take about 1ms, resulting in GNU Radio latency is about 1.256ms. This leads to the application latency is about 4.259ms (Fig. 11). These results show that the time for processing at the host computer dominates the communication bus latency and USRP device latency.

VI. Conclusion

This article breaks down the sources of the latency of whole SDR systems then maps them to host application, kernel processes and actual hardware components. For analytical analysis, all buffers are identified and estimated the contributed delay. These latencies are also quantified by two experimental methods. The first method measures RTT between two SDR systems by using TUN/TAP components with GNU Radio *tunnel* program. The second method measures the hardware latency by estimating the time for successfully transmitting samples from FPGA buffer to host computer. The results show that the total latency of the SDR system is reached to some milliseconds and dominated by the application latency. Although the hardware latency is about half of millisecond, it is still high for current wireless network protocols. We believe our study bring better understanding of the latency in SDR platforms and help correcting the implementations of network protocols for SDR systems.

Acknowledgment

This research was supported by World Class University program funded by the Ministry of Education, Science and Technology through the National Research Foundation of Korea (R31-10100).

References

[1] Ettus Research LLC., “Universal Software Radio Peripherals”, http://www.ettus.com.

[2] GNU Radio., http://gnuradio.org/redmine/projects/gnuradio/wiki

[3] Wikipedia., ”Software Defined Radio”, http://en.wikipedia.org/wiki/Software-defined_radio.

[4] M. Cummings and S. Haruyama., “FPGA in the software radio”, IEEE Communications Magazine, 1999.

[5] “WARP: Wireless open Access Research Platform”, http://warp.rice.edu/trac.

[6] ANSI/IEEE std 802.11 part 11., “Wireless LAN medium access control (MAC) and Physical Layer (PHY) Specification”, IEEE press, 1999.

[7] K. Tan, J.Fang, J.Fang, et al., “SORA: High Performance Software Radio Using General Purpose Multi-core Processors”, 6th USENIX Symposium on Networked Systems Design and Implementation 2009, USENIX, 2009.

[8] T. Schmid. O. Sekkat, and M. Srivastava., “An experimental study of network performance impact of increased latency in SDRs”, WiNTECH’07, 2007.

[9] S. Valentin, H. von Malm, and H. Karl., “Evaluating the gnu software radio platform for wireless testbeds”, Technical Report TR-RT-06-273, 2006.

[10] G. Nychis, T. Hottelier, Z. Yang, S. Seshan and P. Steenkiste., “Enabling Mac protocol implementations on Software-Defined Radios”, Proceedings of the 6th USENIX Symposium on networked Systems Design and Implementation, USENIX, 2009.

[11] Y. Lin, Hyunseok Lee, Yoav Harel, Mark Woh, et al., “SODA: A low-power architecture for software radio”. Proceedings of the 33rd International Symposium on Computer Architecture, ISCA, vol.1, pp. 89-100, 2006.

[12] A. Jow, C. Schurgers, and D. Palmer., “CalRadio: A Portable, Flexible 802.11 Wireless Research Platform”, 1st International Workshop System Evaluation for Mobile Platforms, San Juan, Puerto Rico, 2007.

[13] Wikipedia., “Universal Software Radio Peripheral”, http://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral.

[14] L. Choong., “Multi-channel IEEE 802.15.4 packet capture using software defined radio”, UCLA Networked and Embedded Sensing Lab, 2009.

[15] Ettus Company LLC., “Daughterboard products“, https://www.ettus.com/product/category/Daughterboards.

[16] Universal Hardware Driver., “wiki”, http://ettus-apps.sourcerepo.com/redmine/ettus/projects/uhd/wiki.

[17] TUN/TAP Interface Tutorial., http://backreference.org/2010/03/26/tuntap-interface-tutorial/.

[18] J.C. O’Sullivan, P. Di Francesco, U.K. Anyanwu, L. A. DaSilva, and A. B. MacKenzie., “Multi-hop MAC Implementations for Affordable SDR Hardware”, IEEE Symposia on New Frontiers on Dynamic Spectrum Access Networks (DySPAN). pp. 632-636, 2011.

[19] A. Puschmann, M. Kalil and A. Mitschele-Thiel., “Implementation and Evaluation of a Practical SDR Testbed”, 4th International Conference on Cognitive Radio and Advanced Spectrum Management (CogART), 2011.