Software Radio on Smartphones: Feasible?

Yongtae Park
Korea University
ytpark@korea.ac.kr

Jiung Yu
Samsung Electronics
jiung.yu@samsung.com

JeongGil Ko
Electronics and Telecommunications Research Institute
jeonggil.ko@etri.re.kr

Hyogon Kim
Korea University
hyogon@korea.ac.kr

ABSTRACT

In this work-in-progress paper, we explore the expected impacts, the reality, and the technical issues of making MAC / PHY protocols downloadable softwares (like apps) for smartphones. The notion of software radio on a small number of fixed stations has been investigated for decades, but moving its platform to billions of smartphones and even making it downloadable as an app has never been attempted. We firmly believe that such attempt could open new fronts for the software radio research. Beyond considering its possible impacts and ramifications, we demonstrate through measurements that today's smartphone hardware is already capable enough to support the real-time execution of downloadable MAC/PHY software. It is certainly true for low-speed technologies such as ZigBee, and is close to providing real-time operation of higher-speed technologies like Wi-Fi. In our study, we also find that different application processor architectures lead to different bottlenecks in the MAC/PHY processing chain. This implies that the design of protocols that might be executed in software on general-purpose platforms should factor in the processor architecture like multiple cores and SIMD instruction sets, as it will facilitate the softwarization of wireless communication protocols in the future.

1. INTRODUCTION

In terms of platforms, software defined radios (SDRs) have come through two generations so far. Programmable hardwares like field programmable gate arrays (FPGAs) were used in the first generation [1, 2]. As personal computers (PCs) became powerful enough to provide the computation capability required for real-time signal processing, they opened the second generation [3, 4]. Meanwhile, the proliferation of smartphones in the last 5 years has brought so called the post-PC era. According to Gartner [5], more than 1.8 billion smartphone units were sold worldwide in 2013 alone. Extrapolating, we can expect that tens of billions of smartphones will be used across the globe. Our paper ponders an interesting question arising from this trend: what are the expected impacts, the reality, and the technical challenges in turning smartphones into the third generation platform for SDRs?

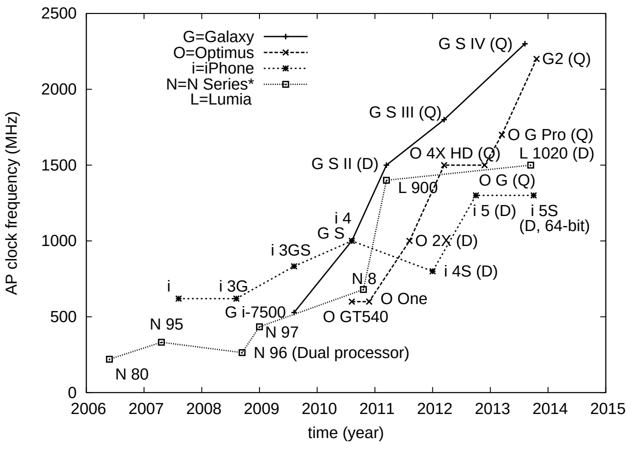

Figure 1: Trends in application processor (AP) speed and number of cores in smartphones

In regard to the first point in the question, we can easily think of multiple ramifications of openly supporting SDR on billions of smartphones. First, when a new MAC/PHY layer communication technology comes along, users will not necessarily have to buy new smartphones like today (*e.g.* the one with 3G, and then with Long Term Evolution (LTE), and then with LTE-Advanced (LTE-A), and so on.). Instead, they will simply download the technology in software and immediately use it, but keep the same hardware. Second, smartphones will be able to support a much wider array of technologies, as they do not need to have separate chipsets and antennas for all of them. Namely, hardware logic can be transformed to software while antennas are shared [6]. Less popular protocols for smartphone users, such as ZigBee, could be supported for those who need it. Older, slower versions of existing protocols could be retired to be supported only in software. Non-standard and experimental technologies could be deployed with less risk as well. By running a range of protocols, smartphones could even serve as protocol translators between devices that do not talk to each other. Third, on the flip side, smartphones could become more compact by saving the real estate by softwarizing protocols that otherwise would need separate chipsets that take up precious space. Fourth, the process of developing and deploying the MAC/PHY protocols can be open to a wider public. We can even imagine a scenario where developers upload a new protocol to a repository to let people download it to install on their devices independently of operating system (OS) updates, like what happens in "app" markets.

The next issue we raise in this paper is how prepared today's smartphones are for this transition of SDR to the third generation platforms. Figure 1 provides a perspective to this question. It shows the trend in the clock rate of the application processor (AP) and the number of cores in top-of-the-line smartphones from some well-known vendors. We can observe that the AP clock rate is steadily increasing, and some (*i.e.*, Samsung Galaxy S4 and LG G2) are already over 2GHz. Even among those behind, the capacity has increased either in the number of cores or in the word width. In particular, the trend towards the multi-core architecture is evident. Dual core was introduced in only 2011, then quad core emerged in 2012, and now octa-cores are available. In essence, the total processing capacity is growing rapidly. We will show later that ordinary smartphones of year 2011 already overtook lower-rate technologies like the IEEE 802.15.4 and some components of the IEEE 802.11a/b in computation. Through novel instrumentation, we will measure up the processing capacity of typical commercial smartphones against the real-time operational requirements of the two IEEE protocol families. And by utilizing three generations of smartphones in the measurements, we will attempt to project the measurement results into future, and predict how the tomorrow's smartphones will fare against them.

Last but not least, we will also discuss the impact of the AP architecture on the performance and its implication on the protocol design, above and beyond the lessons in making the downloadable MAC/PHY apps[1] for smartphones a full-fledged reality.

2. BACKGROUND AND RELATED WORK

Today's smartphone architectures delegate the physical (PHY) and medium access control (MAC) layer computations to the baseband processor (BP), which is interfaced to the radio front end (Figure 2). The radio front converts the over-the-air analog signal to digital form and *vice versa*. The BP performs the remaining PHY layer signal processing and MAC layer computation. Although the dedicated and optimized hardware provides the required performance at a low power budget [7], it has low flexibility. By replacing the inflexible hardware circuitry with software, the software radio can provide multiple benefits: universal connectivity, smaller development cost and time, reuse of existing hardware, and faster upgrade upon protocol evolution [7].

So far, the software radio research has been mostly based on specialized hardware [8, 9, 10, 11]. More recently, however, powerful general-purpose computing platforms such as server class PC [3] or PC with general purpose graphic processing unit (GPGPU) loomed as more flexible alternatives

[1]We use the term "app" in a broader sense here; they may be classified as (library) apps as the downloadable protocols are installed as applications but not as part of OS.

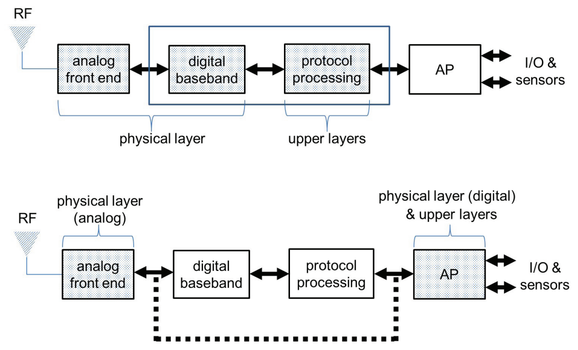

Figure 2: Current architecture (above) and alternative software radio architecture (below)

[4]. Either way, the notion of software radio has been geared toward a small number of fixed stations (*e.g.* base stations). Moving to the newest and the most prevalent platform, *i.e.*, billions of smartphones, has never been attempted.

There is rich literature on software radio. Here, we briefly discuss those that are directly related with the investigation in this paper. Schmid [12] discusses the implementation experiences of the IEEE 802.15.4 module for the GNU Radio. This work shows that the software implementation on the dual Pentium IV platform with 2.8GHz clock and hyper threading performs close enough to the hardware implementation of the 802.15.4 protocol on the Chipcon CC2420 radio used in various mote platforms. Lee *et al.* [7] analyze the workload of the signal processing algorithms in the baseband operation of contemporary wireless networks. The workload characterization is done on two levels, system architecture and individual algorithms like modulation and coding. Kim *et al.* [4] show that to implement SDR the Graphic Processing Unit (GPU) can be programmed to work as a modem that processes the baseband signals with very high parallelism [4]. Dutta *et al.* [13] consider how to build a low-power low-cost SDR hardware platform that could eventually fit on smartphones or sensor nodes. But the work does not consider utilizing existing mobile device hardware to implement the SDR functionalities as we do. Tan *et al.* [3] discuss the design and implementation of a high-performance SDR for WiFi using general purpose multi-core processors. It identifies quantitative requirements such as bus throughput and computation power as well as constraints for real-time operation, in building such a system.

3. SETUP FOR REALITY CHECK

In order to investigate how close today's smartphones are to meet the real-time demand of software MAC/PHY processing, we perform real measurements. Below, we discuss the smartphone platforms, two SDRs, and the configurations used in the investigation.

3.1 Smartphones

We use three smartphone platforms in our instrumentation: Samsung Galaxy S2, Google Nexus, and Samsung Galaxy S4 (LTE-A). They were introduced to the market on February of 2011, November of 2011, and October of 2013, respectively. Roughly speaking, they represent three most recent generations of Android phones. They have been chosen so that we can map the trend in computation power of typical smartphone platforms as measured against the required workload for MAC/PHY processing of the benchmark wireless protocols. Although these phones all run Android OS, the versions are different: Icecream Sandwidth, Gingerbread, and Jellybean, respectively. Nexus and Galaxy S2 use different APs based on the same ARM Cortex A9 architecture: TI OMAP 4460, and Samsung Exynos 4120. Galaxy S4 uses Qualcomm Krait 400, but it runs ARMv7 instructions so the same test program can be used across the three machines. Nexus and Galaxy S2 have two cores, each running at 1.2GHz (*i.e.*, they are 2011 specs; 2012 specs have higher clock rates and more cores). The L1 and L2 cache sizes are also the same, at 32KB and 1MB, respectively. The ARM NEON extension provides a 128-bit single instruction multiple data (SIMD) architecture extension for the ARM Cortex-A series processors. The Galaxy S4 has four cores with 8KB L0 cache, 32KB L1 cache, and 2MB L2 cache.

3.2 Software radios

For our tests, we use publicly available sources for software radio research. For the IEEE 802.15.4, we use the UCLA extension to the GNU Radio [12]. For the IEEE 802.11 a/b, we use the SORA source [3]. The SORA code utilizes the Intel SIMD instructions, so we ported them to use the NEON extension instead on our platforms.

In order to make the software radios into a downloadable software, we use the Android Software Development Kit (SDK) to create the application user interface and memory access routines in Java. The software radios have been ported in C++, and using the Android Native Development Kit (NDK) r7, they were each packaged as a library. Then using the Java Native Interface (JNI), they were connected to the Java routines.

3.3 Emulating radio front

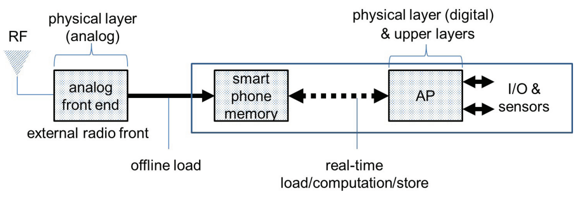

Figure 3: Emulation configuration

The first and the foremost hurdle in testing the software radio on the smartphones is that the radio front is closed. Specifically, the I/O from the radio front cannot be directly tapped by the AP. Instead of employing an external radio front to serve the AP, we emulate the setting instead for ease of experimentation. We put the digitized signal in the memory and read from the memory as if the memory output is the radio front output (Figure 3). The memory throughput is much higher than needed to emulate the radio front.

In order to collect the digitized signal dump for the physical layer packets, we proceed as follows. For the IEEE 802.15.4, we install the GNU Radio 802.15.4 software [12] on a Linux laptop with the Universal Software Peripheral (USRP)[14] Hardware Driver (UHD) that enables it to recognize the USRP as the radio front. For decoding measurement, we need only successfully transmitted packet image, because corrupted packet image is inadequate in measuring the time. For the dump, we use three types of packets: the acknowledgement packet with 0 data bytes, small packets with 9 data bytes, and large packets with 100 bytes of data. Given the packets, the laptop modulates the data, and we capture the signal sample just before it is sent to the USRP. Note that this image is what the receiver-side baseband processor would see just before the digital signal processing commences. Conversely, it would be the output from the sender-side baseband processor just before transmission. Once the dump is collected, we push it into an external SD memory using the Android Debug Bridge (ADB) [15]. Then we load the dump on the memory space of the SDR application on the test smarthone. We pass the dump to JNI C++ function for test decoding. For encoding tests, the output from the same packet input is compared with the packet dump from the Linux laptops above. In case of SORA (the IEEE 802.11 SDR) [3], the signal dump process is done similarly. SORA, however, uses its own radio control board (RCB) to control the radio front. The SORA kit provides the SDK that contains the driver to recognize the RCB. Using the SDK, we create the signal dump to be used for our experiments. In order to check if the signal dump is correct, we perform demodulation over it and compare with the original physical layer packet.

3.4 Real-time requirements

For real-time MAC/PHY processing, we must meet the timing requirements in encoding, decoding, response generation, and event trigger timing [3]. The encoding and the decoding parts should be done in real-time in order to get the line rate. Examples of the response generation are the MAC layer acknowledgement (ACK) to a data frame, or Clear-to-Send (CTS) to a Request-to-Send (RTS), among others. The time given to response generation is Short Inter-Frame Space (SIFS). This hard real-time requirement demands enormous computing power as we will see below, and it is why on today's devices it is usually done in the baseband processor instead of the MAC protocol processor. An example of event trigger timing is Clear Channel Assessment (CCA) for the Carrier Sense Multiple Access (CSMA) protocols. When a packet arrives, it should be noticed by the software radio immediately. If other tasks running at the time of the packet arrival prevent the software radio from noticing and reacting to the packet, it may result in the violation of the given protocol. Also, when the channel is idle, it should be sensed fast enough that the change in the channel state is accurately detected.

4. SOFTWARE ZIGBEE ON SMARTPHONES

Below, we show how the IEEE 802.15.4 SDR application performs on the given smartphone platforms. In particular, we will demonstrate that using only one of the cores, the 802.15.4 MAC/PHY can be executed in real time. Moreover, the smartphones of year 2011 are enough to achieve the realtiem operation.

4.1 Encoding and decoding

As the MAC layer processing overhead is negligible compared with that of the PHY layer, we focus on the PHY processing. The PHY layer PDU (PPDU) has 11 to 31 bytes of

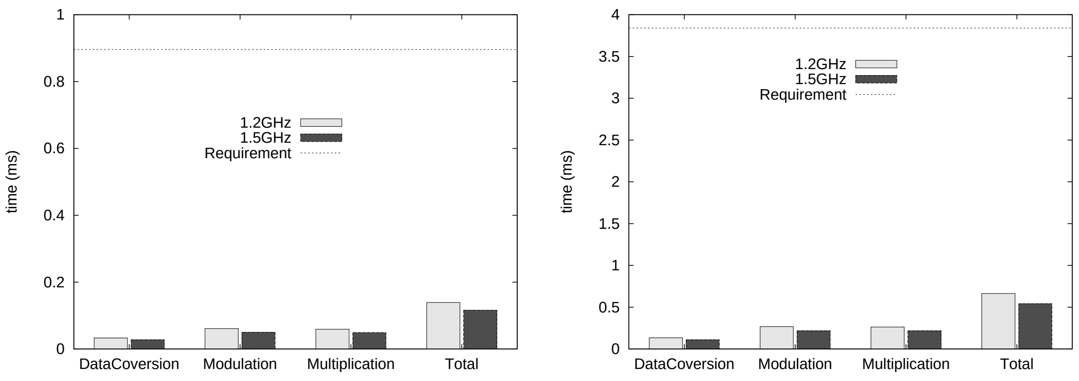

Figure 4: IEEE 802.15.4 encoding performance on Galaxy S2, 1.2GHz and 1.5GHz (overclocked): 9B PHY payload (left), and 100B payload (right)

overhead, and then the data bytes follow. With 100 bytes payload, the deadline for encoding and decoding steps are both 3.84ms. For 9 bytes of data payload, it is 0.896ms. The encoding, decoding, and ACK generation performance on Galaxy S2 are shown in Figures 4 and 5. The numbers are the averages values taken from the result of 10 experiments. We perform the same experiments with two clock speeds, *i.e.*, normal clocking and 25% overclocking. The overclocking could turn out helpful if the normal clocked execution narrowly misses the deadline.

In encoding (Fig. 4), `DataConversion` transforms the bits in the PPDU to symbols and then to chips, the `Modulation` phase performs the OQPSK modulation, and the `Multiplication` phase scales up the modulation output to achieve the transmission gain. The results in Figure 4 clearly shows that realtime 802.15.4 encoding in software is readily handled by a single core of a smartphone three years from the past, not to mention today's and tomorrow's smartphones. The payload size close to the maximum is not a problem, either (Fig. 4, right). Smartphones have surpassed this workload for good.

In decoding (Fig. 5), the `Demodulation` phase performs the OQPSK demodulation, the `IIRFilter` phase cancels noise by removing frequency offset, the `ClockRecovery` phase downsamples and realigns I/Q values, and the `PacketSink` finds the bit patterns like the preamble and the delimiters. Although the 802.15.4 decoding process takes more time than the encoding process and threatens the deadline in the large payload case (Fig. 5, center), it is also within the real-time deadline. With almost twice the cycles budget available on today's smartphones, however, even the large packet would be no match.

4.2 Response

The acknowledgement time t_{ack} in the IEEE 802.15.4 has the following requirement: $t_{TA} \leq t_{ack} \leq t_{TA} + t_{BO}$. Here, t_{TA} is the turnaround time that is 12 symbols long (0.192ms). The t_{BO} is a unit backoff period that is 20 symbols long, so the acknowledgement time is between 0.192ms and 0.512ms. The ACK frame size is 12 bytes, including the PHY header of 6 bytes, PHY payload of 5 bytes, and a single padding byte. Figure 5 (right) shows that the total time to create an ACK frame on Galaxy S2 is even less than t_{TA}, so the ACK frame will be ready by the time the radio is ready to transmit again.

5. SOFTWARE WI-FI ON SMARTPHONES

Unlike low-speed technologies like ZigBee, software WiFi processing requirement is still beyond the computation capacity of today's smartphones. With the example of the IEEE 802.11 SDR, we briefly discuss the issues to run higher-speed MAC/PHY protocols in software on smartphones in real time. In order to minimize the interferences from other processes that might run concurrently with the SDR, we adopted core pinning for the 802.11 SDR application thread(s).

5.1 IEEE 802.11b

The IEEE 802.11 protocols require heavy computation. According to an estimate [3], the 802.11b and the 802.11a require the raw computation power of 10GOPS and 40GOPs, respectively. Table 1 shows the execution times we obtained for the IEEE 802.11b on our platforms, using a single core. With 1KB data payload used in the experiment, the en/decoding and the ACK generation timing requirements are 8.180ms and 0.010ms, respectively. One encouraging result is that the decoding performance can be within the real-time operation bound. The Nexus phone with SIMD processing meets the deadline, and so does S4 SIMD by a large margin. Unfortunately, the execution times for encoding still far exceed the required time. Even with SIMD programming, the encoding time is more than 15-fold higher than the

	Decoding	Encoding	ACK	Android version
Req. (ms)	**8.180**	**8.180**	**0.010**	
Nexus	8.764	815.933	13.341 (N/A)	Icecream Sandwitch
Nexus SIMD	7.609	130.376	2.599 (N/A)	
S2	10.561	903.905	14.139 (0.007)	Gingerbread
S2 SIMD	9.851	128.805	2.511 (0.004)	
S4 SIMD	4.554	35.259	0.574 (0.001)	Jellybean

Table 1: IEEE 802.11b performance (in ms), with 1KB data payload

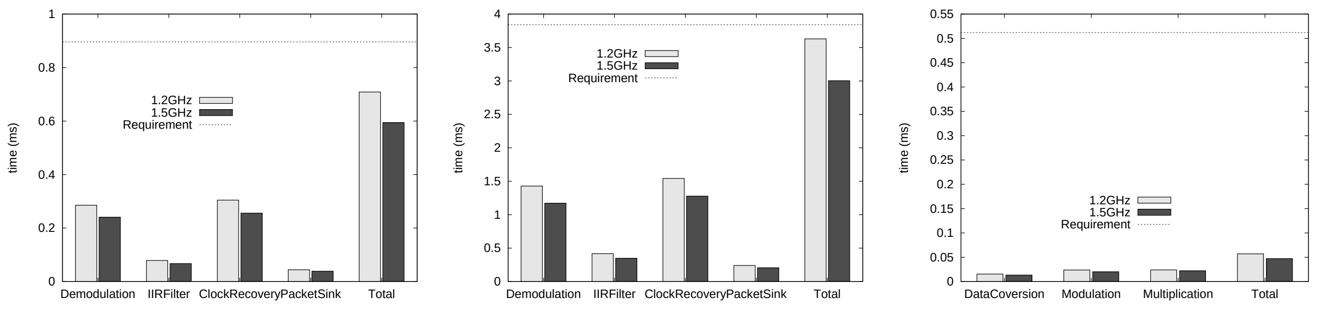

Figure 5: IEEE 802.15.4 decoding performance on Galaxy S2, 1.2GHz and 1.5GHz (overclocked): 9B PHY payload (left), and 100B payload (center); and ACK generation (right)

required deadline. Analyzing the encoding blocks, we find that the finite impulse response (FIR) filter for the `Pulse Shaping` is the culprit. This is because the data has been exploded by `Spreading`. The 802.11b code performs convolution over 160 chips around the one being pulse shaped, and it causes the heavy overhead. However, there is a gleam of hope. Projecting the 3-fold reduction from S2 to S4, we may have the encoding done within time 2-3 years down the road.

The ACK response is not generated in time after the end of the corresponding data frame. SORA also faces the problem for large data frame sizes [3]. The 802.11b SIFS ($10\mu s$) is too short in which to prepare the ACK packet. However, if the destination of the ACK is known, a pre-calculated version can simply be retrieved from memory and transmitted as soon as the data frame integrity check is passed. In the infrastructure mode, the received data comes from the access point with which the mobile is associated, and the MAC address of the access point is indeed known by the mobile. The other fields are frame control, duration/ID, and the CRC. In the frame control, all fields are static for the ACK, and the duration is always 0. As a result, the CRC value can also be precomputed. In case the pre-computed ACK is transmitted, the delay would be well below the SIFS (in parentheses in Table 1; Nexus numbers unobtainable due to coarse measurement time granularity). Again, for S4, we get a 4-fold reduction from S2, so we hope that even real-time ACK generation might be possible in 4-6 years of time.

There is another lesson from Table 1: the use of SIMD is a huge factor. SIMD programming directly impacts the FIR filter, and it reduces the total time by a factor of more than 5 in case of ACK generation, and more than 6 in encoding. As the data width for the SIMD on ARM Cortex is 4 times larger, the improvement should max out at 4. However, the larger improvement is due to the fact that without SIMD, it becomes more likely that other processes can cut in between the instructions for the 802.11b SDR.

5.2 IEEE 802.11a

The computational requirements for the 802.11a protocol are even larger [3]. Table 2 summarizes the test results of running the 802.11a code as an application on two cores (the Viterbi decoder thread uses a separate core as in SORA [3]). We notice the numbers without SIMD programming is nowhere near the requirements. Even with SIMD, they are still far above the requirements. In case of both encoding and decoding, another 2-fold decrease is necessary from S4 numbers. Again, 2- to 3-fold reduction from S2 is encouraging, so we can hope that both encoding and decoding will be done in real-time in 2-3 years time. For ACK generation, we need pre-computation to meet the real-time deadline. However, the 3-fold decrease from S2 to S4 lets us expect *bona fide* real-time operation in 4-6 years in future.

	Decoding	Encoding	ACK	Android version
Req. (ms)	**1.355**	**1.355**	**0.016**	
Nexus	91.547	28.966	0.760 (N/A)	Icecream Sandwitch
Nexus SIMD	4.741	5.717	0.193 (N/A)	
S2	20.331	18.727	0.662 (0.003)	Gingerbread
S2 SIMD	4.594	6.812	0.177 (0.002)	
S4 SIMD	2.078	2.291	0.062 (0.002)	Jellybean

Table 2: IEEE 802.11a performance (in ms)

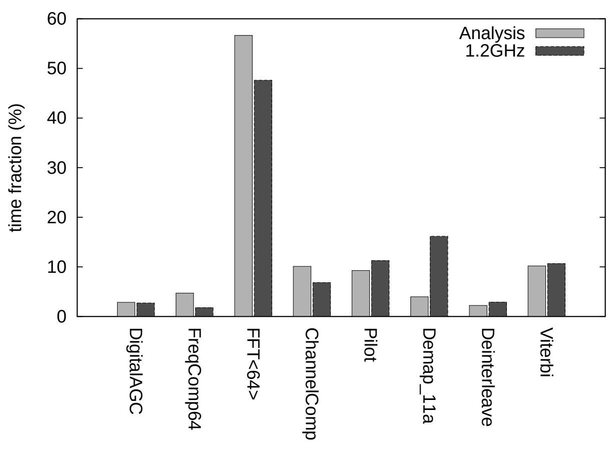

Figure 6: IEEE 802.11a decoding computation breakdown

An important lesson from the 802.11a measurements is that the variation in the SIMD instruction set and register file size can drastically change the bottleneck in the same software radio routines. In order to illustrate this point, we compare the result of running the 802.11a software radio

(MAC/PHY) on Galaxy S2 smartphone with what SORA reports [3]. Figure 6 shows the breakdown of the 802.11a decoding time (encoding shows a similar result so it is omitted for space). As we can see, the `FFT64` takes the largest fraction of time. It is even longer than the Viterbi, which is declared the dominant bottleneck in SORA prototype [3]. The reason is in the number of instructions. First, the SIMD instruction sets supported by Intel and by ARM are different. In particular, there is no vector permutation instruction in the latter, which contributes to the overhead. Second, the simple memory move instructions in the FFT code do not sequentially access the memory and it causes cache misses that affect the two architectures differently. It suggests that there is a complex interplay between the MAC/PHY algorithms and the processor architecture they run on. When we design a protocol (MAC/PHY processing blocks) in future, we might have to be more conscious of general processor architectures on which the protocol might run.

6. DISCUSSION

The software radio for low-speed protocols like IEEE 802.15.4 can already be executed in real-time as a user process if only the application processor could tap the radio front. It clearly signals that the MAC/PHY "app" is a feasible concept in today's smartphones. Considering how the OS is updated in smartphones today, this concept could be crucial for SDRs to avoid the dependency on the OS suppliers. Specifically, for a newly developed or a modified protocol, an app form would allow almost immediate deployment or test, and only on the phones that do need the protocol, not affecting the other users. For example, applications like Internet of Things (IoT) for which lower-speed communication is enough, a MAC/PHY that fits the application better could be custom designed, implemented in software, downloaded as an app, and be run on smartphones.

As the execution time of the software depends on the processor architecture like SIMD instruction sets, architecture-conscious protocol design is desired. We showed that on the smartphones used in this study take much more time for 802.11a in FFT instead of the Viterbi decoder that has been identified as the heaviest processing block in other studies on PCs. Also, as the processor architecture evolves towards multi-core, the protocol design conscious of the multi-core processor architecture would be beneficial. The FFT blocks in the 802.11a protocol for instance, could be executed faster if it could be split into multiple threads. Also, the Viterbi decoding for 802.11a is horribly slow, but Kim *et al.* [4] reports that the Viterbi decoding for 802.16 could be accelerated by a factor of 90 as a result on a multi-core GPGPU platform. It implies that if protocols can be designed to exploit parallelism in software execution, they can become more feasible.

Another issue in SDR is the real-time support from the OS. In future, we will investigate the impact of other concurrent tasks running on the same or other cores.

Finally, the most difficult hurdle in realizing the SDR app approach is the open radio front. That is, the radio front should be directly accessible from the application processor and the memory. One workaround that we can consider today is using an external radio front module plugged into the smartphone USB slot. USB 3.0 theoretically supports up to 5Gbps, so it will be sufficient to support 802.11a/g [3], if not higher-speed technologies. However, we suggest that the smartphone vendors open up the radio front, as the SDR app will allow the smartphones to wield far more extensive communication capability without investing additional chips and space in the device.

Acknowledgements

This work was supported by the Mid-career Researcher Program through NRF grant funded by the MEST (2011-0028892) and also by project number #10041725 from KEIT and the Korean Ministry of Science, ICT and Future Planning.

7. REFERENCES

[1] P. Johnson, "New Research Lab Leads to Unique Radio Receiver," *E-Systems Team*, 5(4), May 1985. http://chordite.com/team.pdf.

[2] P. Hoeher and H. Lang, "Coded-8PSK modem for fixed and mobile satellite services based on DSP," in *Proc. First Int. Workshop on Digital Signal Processing Techniques Applied to Space Communications*, Nov. 1988.

[3] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. Voelker, "Sora: High-Performance Software Radio Using General-Purpose Multi-Core Processors," *Communications of the ACM* (CACM), 54(1), Jan. 2011.

[4] J. Kim, S. Hyeon, and S. Choi, "Impelementation of an SDR System Using Graphics Processing Unit," *IEEE Communications Magazine*, March 2010.

[5] Gartner, "Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013," Nov. 2013. Available at: http://www.gartner.com/newsroom/id/2623415.

[6] S. S. Hong, J. Mehlman, and S. Katti, "Picasso: Flexible RF and Spectrum Slicing," in Proc. of ACM SIGCOMM, 2012.

[7] H. Lee, *A Baseband Processor for Software Defined Radio Terminals*, Ph.D. Thesis, U. of Michigan, 2007.

[8] Texas Instruments, "Small Form Factor SDR Development Platform," *Product Bulletin*, http://www.ti.com/lit/ml/sprt434a/sprt434a.pdf.

[9] WARP: Wireless Open Access Research Platform, http://warp.rice.edu/trac.

[10] M. Cummings and S. Haruyama, "FPGA in the Software Radio," *IEEE Communications Magazine*, 1999.

[11] J. Glossner, E. Hokenek, and M. Moudgill, "The Sandbridge Sandblaster Communication Processor," in *Proceedings of 3rd Workshop on application Specific Processors*, 2004.

[12] T. Schmid, "GNU Radio 802.15.4 En- and Decoding," unpublished document and source, 2006. Available at: http://nesl.ee.ucla.edu/document/show/282.

[13] P. Dutta, Y.-S. Kuo, A. Ledeczi, T. Schmid, and P. Volgyesi, "Putting the Software Radio on a Low-Calorie Diet," in *Proceedings of HotNets*, Oct. 2010.

[14] Ettus Research, LLC. http://www.ettus.com/.

[15] Android Debug Bridge, http://developer.android.com/guide/developing/tools/adb.html.