Resource Management with Real-Time Complexity Monitoring in Software-Defined Radios

Joseph D. Gaeddert and Jeffrey H. Reed
Wireless@Virginia Tech
Bradley Department of Electrical and Computer Engineering, Virginia Tech
432 Durham Hall, MS 0350, Blacksburg, VA 24061, USA
E-mail: jgaedder@vt.edu

***Abstract*—Software-defined radios (SDR) are typically realized as the deployment of modular digital signal processing blocks on a variety of platforms and promise significant enhancements over traditional radio designs in terms of component re-use and design scalability. Despite their apparent benefits, however, software-defined radios tend to consume more power and provide less throughput than their hardware counterparts (e.g. application-specific integrated circuits and field programmable gate arrays). As a result, SDR platforms have struggled to compete with many legacy systems, particularly as the demand for data throughput on wireless infrastructure increases. This significant performance gap can be mitigated, however, by exploiting the reconfigurable nature of software itself and by compensating for opportunistic channel conditions by pruning unnecessary processing. Our previous work demonstrated that the receiver could significantly reduce its computational complexity while maintaining an equivalent error probability in Rayleigh fading channels simply by choosing a more strategic modulation-/coding-scheme pair at only a slight degradation to its spectral efficiency.**

This paper extends our previous work by formalizing computational complexity as an analytical model for monitoring the processing load in digital hardware. This model is used to adapt link-level parameters on reconfigurable software radio platforms where processing bandwidth is a discerning factor for data throughput. Furthermore, the results are extended by incorporating additional physical-layer considerations into the analysis. Finally, the system is validated through the assessment of real-time software simulation as well as over-the-air reconfiguration in a controlled wireless environment.

I. Introduction

The wireless industry has an ever-growing appetite for data throughput, and with it comes the need to adapt to quickly-changing standards. Software radio is heralded by some in the wireless engineering community as the future of standards implementation, however this mentality might be somewhat short-sighted. Reconfigurability, however advantageous in theory, comes at the expense of increased power consumption and hardware costs. Software-defined radios (SDR), despite all their proposed benefits, consume considerably more power than their hardware counterparts, such as application-specific integrated circuits and field programmable gate arrays. The energy efficiency lost through the flexibility of baseband processing on a reconfigurable platform is a severe limitation to its computational capacity and has considerable implications to its physical size, weight, and battery life [1]. Furthermore, the steady increase of hardware capabilities is easily overwhelmed by the escalating demand for high-quality bandwidth on mobile wireless devices. As such, the performance of software-defined radio platforms will continue to lag behind conventional hardware designs.[1]

As shown in [2], [3], real-time power monitoring can be accomplished by modeling processing load for streaming multimedia systems. Furthermore, [4] demonstrated a significant resource reduction by selectively reducing processing to meet a specific target service quality. In our previous work we demonstrated the strong relationship between channel conditions and computational complexity required by the receiver, specifically through the adaptation of forward error-correction (FEC) coding in conjunction with the scheme used to modulate the underlying data [5]. While the tradeoffs between received signal strength (relative to the noise level) to error rate performance is well known within the wireless engineering community, its impact on computational resources is not. This is in part due to the complexities of modeling complex algorithms on various hardware devices and is significantly dependent upon the algorithm, the physical charactersitcs of the platform, and the implementation. Professional signal processing engineers use sophisticated benchmarking and profiling tools to eliminate any and every unnecessary instruction in the algorithm to ensure its performance is streamlined and as near to optimum as possible. While a useful design approach, it is impractical to apply it to the modern model-based software design in which there exists a strict disconnect between the platform-independent and platform-specific models.

In this paper we propose the use of online processor complexity monitoring in order to specifically improve the computational efficiency of the receiver in a dynamic wireless environment. Section II describes the resouce management model used in our analysis, Section III explains our exper-

[1]In this paper *reprogrammable* refers to the platform's ability to run different waveform adaptations on a single piece of hardware. *Reconfigurability*, however, refers to its ability to swap out components dynamically, perhaps on a per-packet basis. In this regard, field-programmable gate arrays (FPGAs) are indeed reprogrammable as they have the ability to be flashed with different images, yet because they cannot be dynamically adapted while the image is running, they are not reconfigurable. We acknowledge that a number of research efforts have been put into studying dynamic reconfiguration of FPGAs; at the time this paper was written, however, their reconfigurable nature was not readily available and did not match the performance to that of software platforms. Furthermore, while it is possible to switch *between* modules on an FPGA as it is running, the modules themselves are otherwise fixed.

imental setup, Section IV summarizes the results found in our study, and Section V gives our final conclusions on our analysis.

II. Resource Management in SDR

Radios in general require a variety of resources in order to operate, such as power, energy, and spectrum. Software-defined radios in particular require not only more resources than their hardware-defined counterparts, but a different set of resources altogether. Specifically, computational bandwidth and flash memory are precious commodities for SDR. Power consumption on a software-defined receiver is typically higher due to its more demanding processing requirements. As we have demonstrated in [5] through extensive base-band simulation, receivers need to work harder to recover signals which are weaker, noisier, and in general more corrupted. Traditional radio designs implement standards in hardware; the physical layer is designed to operate under the predicted worst-case scenarios and has little flexibility for on-the-fly modification to adapt to changing wireless conditions aside from switching between modulation and forward error-correction coding schemes. While these adaptations play a signficant role in link throughput, data reliability, and energy consumption, other common physical-layer parameters such as filter length, number of equalizer taps, rake fingers, and image-rejection filter length are often left untouched by the receiver and viewed as pre-specified. Neglecting these dimensions of communications signal processing during the design process results in an acutely sub-optimal solution. Just as any embedded software development must consider resource consumption on its host platform, so must SDR in this regard.

A. Performance Metrics

Understanding the dynamics behind algorithms and the resources they consume is necessary in gaining back what SDR loses through flexilibity. We propose the use of computational channel complexity, $\mathcal{K}$, as a performance metric for comparing receiver designs. Computational channel complexity is simply the number of clock cycles at the receiver which are required to properly synchronize and decode a single bit of information over a communications link. It is therefore necessary to monitor waveform complexity at the receiver in order to gain insight as to how well the receiver structure performs. We model the processor's clock cycles as a scarce commodity and limited resource for facilitating communications. $\mathcal{K}$ has units clock cycles per bit and can be measured several ways. For most platforms, processor usage can be easily measured on certain running processes as a strict percentage of its maximum load. As such, a processor with a master clock frequency of F_p cycles/second operating at r_p percent of its maximum while running a receiver capable of decoding η bits per second per Hz has a processing complexity of

$$\mathcal{K} = F_p r_p / \eta \tag{1}$$

While this quantity gives an indication of peak computational complexity, it does not thoroughly describe the actual throughput of the link given its channel quality. As demonstrated in our previous work, channel quality has significant implications on the workload required by the receiver. Throughput, in this case, is a measure of properly-received data (often referred to as goodput), and is computed as

$$\zeta(\gamma) = \eta\Big(1 - P_e(\gamma)\Big) \tag{2}$$

where again η, is the theoretical spectral efficiency of the transmission scheme (e.g. modulation/coding scheme pair with framing overhead) and P_e is the packet error rate at the receiver under an instantaneous signal-to-noise ratio (SNR), γ. Both η and P_e are strongly affected by the choice of radio transmission scheme, as is the resulting receiver complexity. This normalized quantity describes the actual amount of data being conveyed over the wireless channel. The quantity in (2) is theoretically bounded by γ and is well-known in information theory. Typically channel capacity is referenced not in terms of the ratio of signal energy to noise (E_s/N_0), but of bit energy to noise (E_b/N_0). The use of E_b/N_0 is viewed as a fair comparison of different transmission schemes as it incorporates the rate of the transmission into its analysis of signal strength. However we use E_s/N_0 because the receiver's signal-to-noise ratio incorporates its ability to synchronize the entire physical-layer frame, including detection, acquisition, and tracking. The ratio E_b/N_0 is really only useful once the receiver has synchronized, ignoring a very important and practical component of wireless communication.

For a fair comparison of transmission schemes, the complexity of the receiver must be normalized to the actual throughput. Incorporating (1) and (2) gives

$$\bar{\mathcal{K}}(\gamma) = \mathcal{K}/\zeta(\gamma) \tag{3}$$

which can be undersood as the average number of clock cycles to decode a single bit of data through a communications link, while accounting for erroneous data.

B. Measurement and Framework

It is fairly straightforward to benchmark and profile signal processing algorithms running on a general purpose processor, and the open-source software community has provided a number of tools to accomplish these tasks. In regards to processor usage, POSIX kernels offer several possibilities for real-time monitoring of CPU and memory usage and provide a programming interface through the `rusage` structure in the standard `sys/resource.h` library header. The `getrusage()` method reports resource usage for a specific process running on the system, having the capability to estimate not only the amount of time in which the processor has spent executing instructions but other useful metrics such as memory footprint and number of page faults. This interface provides an invaluable window into the processor's core, granting the radio access to information which it can use to improve system performance.

While this interface provides us the ability to monitor several resource metrics, we only consider computational complexity of running the algorithm on the processor of

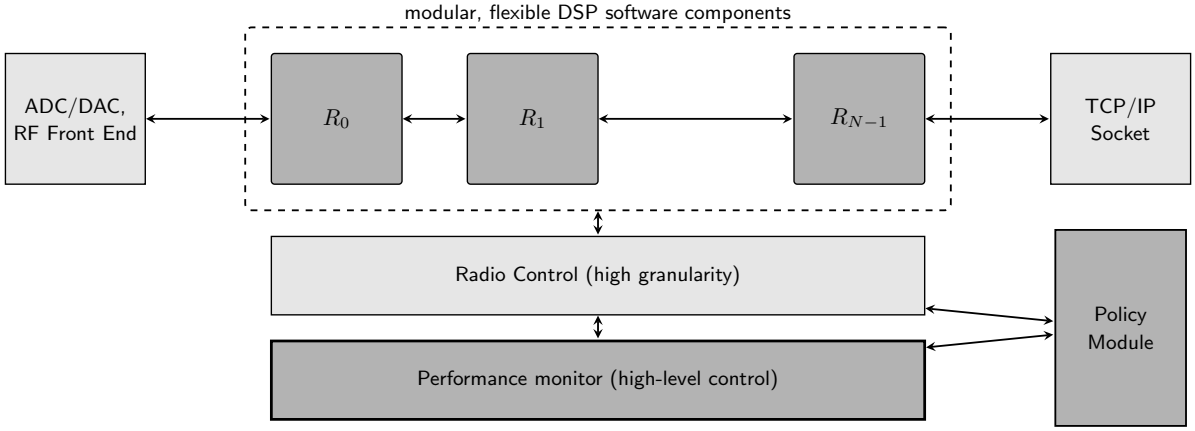


Fig. 1. Resource management framework

interest. This choice was made out of simplicity, leaving future development efforts to investigate the effects of considering additional resources. The complexity of the algorithm is directly proportional to the processor load r_p, which is simply the relative time the processor spent running the algorithm (T_a) divided by the total amount of time it spent in execution (T_e), viz

$$r_p = T_p/T_e \tag{4}$$

The quantities T_p and T_e are relatively easy to compute, and emperical evidence on our target platform shows that these resource usage measurements are both accurate and consistent. Furthermore, invoking these interfaces adds minimal if not negligible overhead to the radio's performance during its operation. This implies that a performance monitor can periodically probe the state of the radio platform and continually observe its resource usage. Figure 1 depicts the proposed real-time resource management framework for monitoring performance of the system on a software platform. The parameterized resources $\{R_0, R_1, \ldots, R_{N-1}\}$ are adjusted by the *Radio Control* module which provides an interface for setting the radio's physical layer parameters (filter parameters, modulation scheme, transmit power, etc.) while the *Performance Monitor* evaluates the online system performance. The purpose of the *Policy Module* is to ensure that the parameterized solution set is valid for the regulated protocol. As the *Performance Monitor* can observe the resource usage of the system as it runs, the radio can operate in a mode appropriate to the conditions of the channel: running in an enhanced mode when conditions are poor, and scaling back when conditions are good.

III. Experimentation

Consider a scenario in which two software radios need to establish a communications link in a wireless network. These nodes need to negotiate a protocol which achieves the highest data rate possible without exceeding the capabilities of either node. For most protocols, the burden of computation is much higher at the receiving end of the link and is therefore the limiting factor. Furthermore heterogenous networks are likely to contain nodes of vastly different capabilities; some with higher computational bandwidth than others. Should they just select one of a pre-determined set of protocols, it is likely that if their capabilities overlap, their solution will be sub-optimal, and the performance poor. In our previous work, we demonstrated that simply choosing an appropriate set of modulation and FEC coding scheme pairs can result in a significant performance improvement. Here, we validate these results through over-the-air data transfer in a packet radio network. While a number of other radio parameters significantly affects the computational complexity of the receiver, we limit our analysis to simply modulation and coding schemes as to be consistent with our previous analysis, and leave other parameters–e.g. number of equalizer taps, filter length–to another paper.

A. Framing structure

In order to validate the results of our previous work, a simple physical-layer wireless protocol was developed in which narrowband packets were transmitted between two software-defined radio platforms. The structure for framing the packets was designed to be as versatile as possible, allowing for fast acquisition, reliable detection, variable payload lengths, coherent demodulation of any number of modulation schemes, and decoding of layered, interleaved FEC schemes.

The framing structure consists of six basic components, as depicted by Figure 2. The ramp up, phasing pattern, and P/N sequence are together known as the physical layer convergence procedure and are used for fast acquisition of the frame at the receiver through packet detection, automatic gain control (AGC) lock, carrier phase-locked loop (PLL) lock, and symbol timing PLL lock. The ramp up and preabmle phasing sequences are a BPSK pattern which flips phase for each transmitted symbol (`+1,-1,+1,-1,...`). This sequence serves several purposes but primarily to help the receiver's symbol synchronization control loop lock onto the proper timing phase. The ramping nature of the beginning of the frame gracefully increases the output signal level to avoid "key clicking" and reduce spectral side-lobes in the transmitted

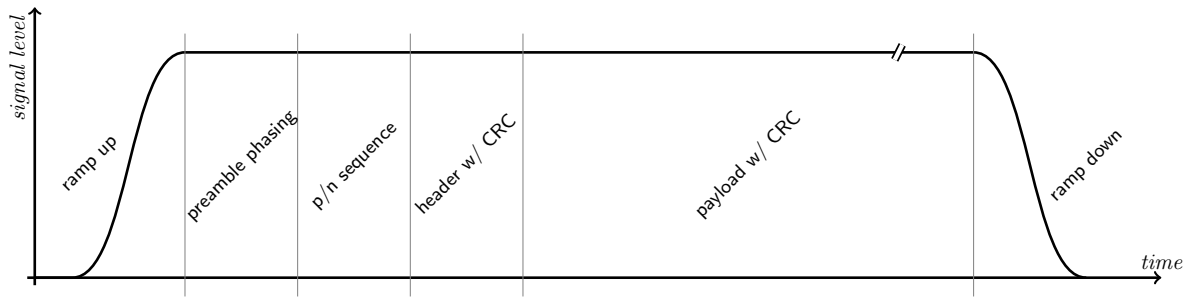


Fig. 2. Framing structure used in over-the-air experimentation. The payload length is variable and is typically much longer than the preamble.

signal. Furthermore, it allows the receiver's automatic gain control unit to lock on to the incoming signal, preventing sharp amplitude variations in its output. The P/N sequence is a 63-bit m-sequence (maximal length) exhibiting good auto- and cross-correlation properties. This sequence aligns the synchronizers to the remainder of the frame, indicating when to start receiving and decoding the frame header, as well as indicating the coherent phase of the received signal. At this point it is assumed that the receiver's AGC, carrier PLL, and timing PLL all have locked to the signal at which time the bandwidths of each are reduced to a suitable tracking mode. The header consists of a several data bytes describing to the receiver how to decode the payload. It adds a minimal amount of overhead to the frame as a whole and permits the receiver to reconfigure itself on-the-fly for each packet it detects. It includes a 32-bit cyclic redundancy check to validate data integrity and is encoded using a simple $2/3$-rate Hamming(12,8) block code and modulated using BPSK. The payload is the core of the frame, containing the raw data to be transferred across the link. Its length is variable depending upon the number of raw payload bits, forward error-correction used (if any), and modulation scheme used. Before any error-correction encoding is applied, a 32-bit cyclic redundancy check is appended to the raw data to validate that the frame was properly received. Finally, the ramp down sequence serves to gracefully lower the transmitted power of the signal.

B. Transceiver description

A fully-reconfigurable software transceiver was implemented in the C programming language capable of synchronizing to packets transmitted over a wireless channel. The receiver consists of the following modules: automatic gain control with squelch to disable unnecessary processing when the signal strength is low, received signal strength and signal-to-noise radio estimation, numerically-controlled oscillator and second-order PLL for carrier synchronization, multirate decimator for symbol timing recovery [6], frame lock detection, variable-level demodulation, block de-interleaver, forward error-correction decoder, and cyclic redundancy check validator.

C. Experimental Setup

The goal of the experiment was to achieve a maximum throughput (ζ) given an instantaneous signal-to-noise ratio (γ) by switching between available modulation and forward error-correction schemes from within a set. Two different sets were chosen: one on the basis of maximizing spectral efficiency, and the other on the basis of minimizing computational complexity. The first set, $\mathcal{S}_\eta$ (designed to maximize spectrum efficiency), contains typical $r = 1/2$ and $r = 1/3$ convolutional codes with a constraint length $K = 9$ and punctured at rates ranging from $2/3$ to $7/8$. The second set, $\mathcal{S}_\mathcal{K}$ (designed to minimize computational complexity), uses either no error-correction (uncoded) or only a computationally efficient Hamming$(12, 8)$ $2/3$-rate block code. The two sets are depicted in Table I along with their average computational loads measured in the number of processor clock cycles (in thousands) required to decode a single bit of data through the entire receiver chain. Notice that their spectral efficiencies have nearly the same range (from 0.6667 b/s/Hz at a minimum to about 5 b/s/Hz at a maximum), yet have starkly different receiver complexities.

Frames were transmitted over a wireless link in a laboratory environment with varying transmit powers such that the receiver's signal-to-noise ratio deviated from approximately -2 dB to 23 dB. The transmitter and receivers ran on independent Intel Pentium processors, each connected to a USRP [7] and operating at a symbol rate of 225 kHz with a payload of 1024 bits (128 bytes). The receiver made an estimate $\hat{\gamma}$ of each frame's signal-to-noise ratio using a combination of the received signal strength relative to its pre-characterized noise level and the variance of its demodulator error vector magnitude. Statistics were gathered for the data throughput and computational complexity under each modulation and coding scheme pair from each of the two sets depicted in Table I. These complexity measurements were conducted by invoking `getrusage()` as described in Section II-B and using (1) and (4). It is important to recognize that the clock cycle measurements in Table I incorporate the entire receiver including filtering, gain control, frame detection, carrier frequency/phase recovery, symbol timing synchronization, decimation, demodulation, and forward error-correction decoding, while our previous work only incorporated the demodulator and FEC decoder.

TABLE I
AVAILABLE MODULATION/FEC SCHEME PAIRS, BROKEN INTO 2 SETS, $\mathcal{S}_\eta$ AND $\mathcal{S}_\mathcal{K}$

Modulation scheme	FEC scheme	η, [b/s/Hz]	$\mathcal{K}$, [k-cycles/bit]
	$\mathcal{S}_\eta$, spectrum efficiency-driven set		
QPSK	conv. $r = 1/3$, $K = 9$	0.667	12.512
QPSK	conv. $r = 2/3$, $K = 9$	1.333	8.135
QPSK	conv. $r = 3/4$, $K = 9$	1.500	7.568
QPSK	conv. $r = 7/8$, $K = 9$	1.750	7.064
16-QAM	conv. $r = 2/3$, $K = 9$	2.667	6.138
16-QAM	conv. $r = 3/4$, $K = 9$	3.000	5.733
16-QAM	conv. $r = 4/5$, $K = 9$	3.200	5.753
16-QAM	conv. $r = 7/8$, $K = 9$	3.500	5.281
64-QAM	conv. $r = 2/3$, $K = 9$	4.000	5.254
64-QAM	conv. $r = 3/4$, $K = 9$	4.500	4.975
64-QAM	conv. $r = 4/5$, $K = 9$	4.800	4.840
64-QAM	conv. $r = 7/8$, $K = 9$	5.250	4.831
	$\mathcal{S}_\mathcal{K}$, computational complexity-driven set		
BPSK	Hamming $(12, 8)$	0.667	7.874
QPSK	Hamming $(12, 8)$	1.333	3.239
QPSK	uncoded	2.000	3.013
8-PSK	uncoded	3.000	2.264
16-QAM	uncoded	4.000	1.938
32-QAM	uncoded	5.000	1.474

IV. RESULTS

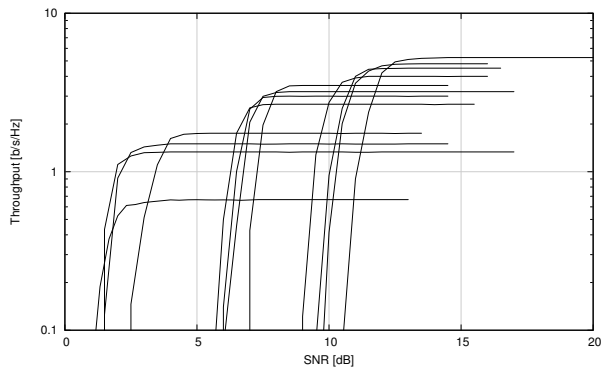
Figure 3 depicts the measured throughputs for the two modulation/coding scheme sets measured over the air in a laboratory environment, computed using (2). As the received signal level degrades, the packet error rate approaches unity, and therefore the throughput approaches zero. As expected, this transition occurs sharply with γ such that an SNR threshold is required to achieve any suitable throughput. This threshold varies for each modulation/coding scheme pair, as does the actual peak throughput.

In an adaptively switching protocol, the radios choose the pair with the highest throughput for a given γ in order to maximize the total throughput for the link. To compare, the peak throughput for each set–the value riding along the crest of Figures 3(a) and 3(b)–is plotted in Figure 4. This shows the maximum throughput possible for the system given a specific channel condition. Finally, Figure 5 shows the computational complexity $\bar{K}$ of the two sets, as computed by (3).

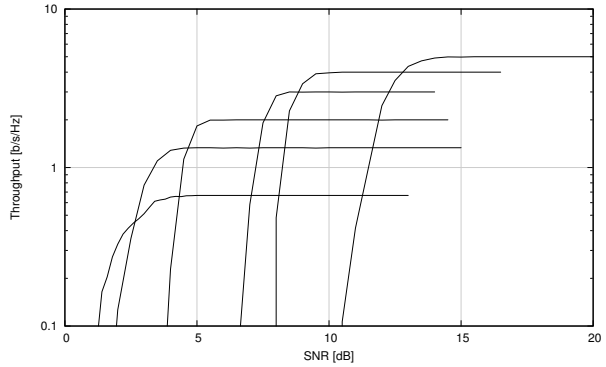
As expected, $\mathcal{S}_\eta$ has typically a higher spectrum efficiency than $\mathcal{S}_\mathcal{K}$, however for certain values of γ it is actually slightly lower. This is demonstrated in Figure 4 near $\gamma = 6$dB and again near $\gamma = 10$dB where the computational complexity-driven set exhibits a slightly higher throughput than that of $\mathcal{S}_\eta$. This discrepancy can be attributed to the synchronizer's inability to lock onto signals with moderate carrier phase and frequency offsets as well as timing phase offsets. While simulation suggests that the packet error rate should be lower for certain modulation schemes, the receiver is unable to extract an adequate channel phase estimate due to noise and therefore cannot establish a carrier lock. This results in a constellation with a gross carrier phase offset and thus a higher bit error-rate than theory under ideal carrier recovery assumptions. In a practical sense, the limiting factor of the physical layer is the distance between signal points as they are responsible for proper demodulation and proper carrier recovery. As a result, the errors introduced by increasing the modulation scheme cannot easily be recovered by using forward error correction.

Clearly the introduction of convolutional decoding increases the number of computations required at the receiver. This is a well-known phenomenon and makes a significant impact on the receiver's computational complexity. Not only does using a stronger forward error-correction scheme typically increase computational requirements of the reciever by running the algorithm alone, but decreasing the rate of the code also increases the length of the frame. This, in turn, requires the receiver to run more filtering and gain operations which directly impact its computational complexity. The processing complexity of set $\mathcal{S}_\eta$ is roughly three times higher than that of set $\mathcal{S}_\mathcal{K}$ at the highest signal-to-noise ratio; this suggests that the receiver could actually increase its spectrum bandwidth running at several times its nominal speed, increasing its data rate. Alternatively, the same receiver could be deployed on a more primitive piece of hardware, allowing for nearly the same throughput to be achieved on nodes without higher capabilities. In any case, it suggests a higher available bandwidth or lower energy consumption at the receiver.

For very low signal-to-noise ratios (less than 3dB) the



(a) $\mathcal{S}_\eta$, spectrum efficiency-driven set



(b) $\mathcal{S}_\mathcal{K}$, computational complexity-driven set

Fig. 3. Performance of adaptive switched modulation/coding scheme sets, average throughput, $\zeta(\gamma)$, measured in an over-the-air, laboratory test environment

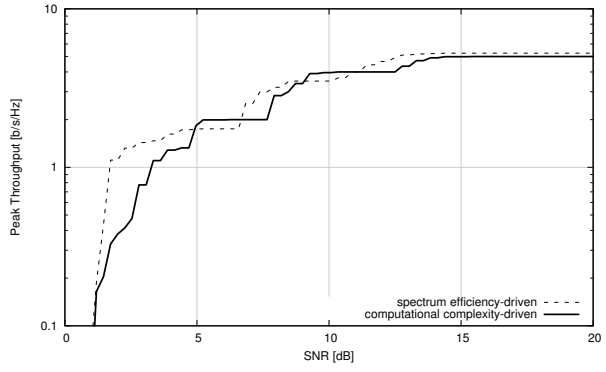


Fig. 4. Peak throughput $\zeta(\gamma)$

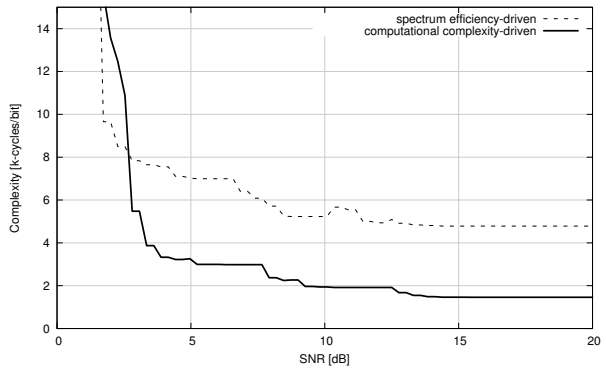


Fig. 5. Computational complexity, $\mathcal{K}$, vs. signal to noise ratio, γ

spectrum efficiency-driven set actually out-performs the computational complexity-driven set; this is because the receiver's throughput is so low that the denominator in (3) approaches zero causing $\bar{\mathcal{K}}$ to explode. However, this is only an issue for low values of γ in which both sets suffer significantly to channel degradations.

V. Conclusions

In this paper we have demonstrated the advantages of monitoring resource consumption of software-defined radio platforms. Our results from [5] have been confirmed with an over-the-air implementation of an adaptive radio capable of switching modulation and coding schemes, saving the host receiver valuable computational bandwidth while sacrificing only a moderate spectral efficiency. We expect to see an even greater performance increase by extending the analysis to encompass the full receiver architecture, and allowing the radio to adjust nearly all aspects of its receiver.

References

[1] M. Uhm, "The Power of Software Defined Radio: A Holistic Approach to Designing SDRs for Power," in *COTS Journal*, 1 2007.

[2] C.-H. Lien, Y.-W. Bai, and M.-B. Lin, "Estimation by Software for the Power Consumption of Streaming-Media Servers," *IEEE Transactions of Instrumentation and Measurements*, vol. 56, no. 5, pp. 1859–1870, October 2007.

[3] Y. Hu, Q. Li, and C.-C. J. Kuo, "Run-time Power Consumption Modeling for Embedded Multimedia Systems," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005.

[4] S. M. Yardi, M. S. Hsiao, T. L. Martin, and D. S. Ha, "Quality-Driven Proactive Computation Elimination for Power-Aware Multimedia Processing," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2005.

[5] J. D. Gaeddert and J. H. Reed, "Leveraging Flexibility in Reconfigurable Baseband Processors for Resource Management in Software-Defined and Cognitive Radios," in *Software-Defined Radio Forum*, December 2009.

[6] frederic j. harris and Michael Rice, "Multirate Digital Filters for Symbol Timing Synchronization in Software Defined Radios," *IEEE Journal on Selected Areas of Communications*, vol. 19, no. 12, pp. 2346–2357, December 2001.

[7] (2010, July) Ettus Research LLC. [Online]. Available: http://www.ettus.com/