Malm, Nicolas; Ruttik, Kalle; Tirkkonen, Olav

Latency-Aware Power Management in Software-Defined Radios

Published in:
Journal of Electrical and Computer Engineering

DOI:
10.1155/2020/1854826

Published: 01/01/2020

Document Version
Publisher's PDF, also known as Version of record

Hindawi
Journal of Electrical and Computer Engineering
Volume 2020, Article ID 1854826, 19 pages
https://doi.org/10.1155/2020/1854826

Research Article

Latency-Aware Power Management in Software-Defined Radios

Nicolas Malm, Kalle Ruttik, and Olav Tirkkonen

Department of Communications and Networking, Aalto University, Maarintie 8, 02150 Espoo, Finland

Correspondence should be addressed to Nicolas Malm; nicolas.malm@aalto.fi

Received 5 June 2019; Revised 20 December 2019; Accepted 28 December 2019; Published 11 February 2020

Academic Editor: Fabio Massaro

Cloud computing provides benefits in terms of equipment consolidation and power savings from higher utilization for virtualizable software. Cellular communication software faces challenges in cloud computing platforms. BSs create a specific load profile that differs from traditional cloud service loads. Cellular communication system implementations have real-time deadlines with fixed, periodic latency requirements. In this paper, we assess the suitability of an unmodified Ubuntu Linux OS running on a commodity server to operate latency-critical software using a 4G LTE BS software-defined radio implementation. Scaling of the CPU clock frequency is shown to be feasible without excessive impact on the platform's ability to meet the 4 ms processing delay requirement imposed by the LTE standard. Measurements show the relationship between the processor's operating frequency and the number of missed subframe processing deadlines to be nonlinear. The results obtained also indicate that a high computational capacity does not suffice to ensure satisfactory operation since fronthaul processing overhead can limit achievable performance. Use of offload-capable network interface cards is studied as a potential remedy.

1. Introduction

Evolution of telecommunication systems is directed by the intersection of demand and available technical solutions. BS design evolution reflects how Moore's law has enabled increased platform flexibility. In early BSs, specialized functions were implemented by discrete elements. Over time, discrete solutions were replaced by specialized ASICs. More recently, the rigidity of ASICs was eschewed in favor of FPGA- and DSP-based platforms. Nowadays, BS processing is within the reach of general purpose processors (GPPs) [1–3].

Increased use of software has enabled greater consolidation of logical functions into physical devices through the use of network function virtualization (NFV), software-defined networking (SDN), and software-defined radio (SDR). Consequently, this trend has lead to investigation of the feasibility of replacing specialized hardware by commodity servers with general purpose processors (GPPs) and even PCs. In this paper, the suitability of cloud servers to provide cellular radio access processing, known as cloud radio access network (C-RAN), is assessed.

General purpose servers aim to provide reasonable throughput for a wide range of tasks and by definition are not optimized for any particular one of them. In order to serve the needs of cellular systems, general purpose computing platforms must be adapted to satisfactorily run latency-sensitive cellular communication applications while still retaining the flexibility inherent in a software-based design.

By porting radio access functionality to a cloud platform, cellular system development can be set onto the same trajectory as other cloud-based services. Virtualization enables consolidation of computing hardware, reduces system cost, provides power savings, and increases flexibility [4]. However, it comes at the expense of process isolation and reduced predictability. Very stringent processing delay requirements in cellular base stations (BSs) may make centralization impractical. In order to remedy this problem, the trend is to migrate the C-RAN computation towards the edge of the network into edge clouds [5, 6].

For C-RAN to realise effective coexistence of multiple BSs, the implementation must cope with the unpredictable delays and variable processing power availability of general purpose server environments. Achieving this aim calls for

the creation of SDR BS implementations and system tests to outline the limits of current server platforms in order to guide their evolution.

In addition to tight latency and timing requirements, energy efficiency has gained importance as a cellular system metric. Global energy expenditure of the ICT industry represents 2% of global consumption and is projected to reach 3% by 2020 [7]. Meeting both the ever growing and more varied needs for wireless communication, along with acceptable levels of energy consumption, requires solutions enabling power management to take into account the latency bounds inherent to real-time cellular systems. In particular, this study focuses on improving power management in a C-RAN type shared computational environment with latency constraints.

In this paper, we describe the specific computational requirements of a cellular BS and what constraints it imposes on a cloud platform. To serve this computational load takes not only software adaptation but also suitable hardware. The suitability of current commodity server platforms is investigated using a 4G cellular system BS running on an Intel server platform. Assessing the suitability of a general purpose Linux-based operating system (OS) constitutes a prerequisite to creating a suitable C-RAN to exploit the consolidation benefits of cloud technologies. Improvement opportunities are outlined based on the results obtained.

This article is organized into nine parts. The structure and functionality of a typical BS are presented in Section 2. Related work is surveyed in Section 3. Sections 4 and 5 describe the pertinent BS features and their implementation in the agile radio framework (ARF) SDR platform. In Sections 6, 7, and 8, the performance model, experimental setup, and the results obtained therewith are presented. Section 9 concludes this article.

2. Base Station Evolution into C-RAN

A cellular BS generates load that is characterized by high computational requirements along with stringent periodic completion deadlines. Because of these requirements, BSs are usually implemented in dedicated hardware with a real-time OS.

System specification evolution requires BS platform flexibility. BS adaptability has progressed from software-controlled hardware-defined radios (HW) [8], through accelerator-based GPP designs [9, 10] to full software physical layer software-defined radio (SDR) implementations [8]. Each evolutionary stage provides more run-time configurability than the previous.

In a cloud environment, it is desirable to use general purpose hardware and virtual machines executing non-real-time OSs. Cloud-based SDRs would be implemented as an application process that can be created with libraries, toolsets, and frameworks employed in general software development. Implementation of BSs in centralized servers is called cloud radio access network (C-RAN). C-RAN systems are usually split up into remote radio head (RRH) and baseband unit (BBU). These units are connected over a fronthaul link. The most efficient way to split functionality between BBU and RRH remains an open question. One of the issues is in which system component to place physical and MAC layer processing. This paper describes a centralized approach to C-RAN. RRHs receive modulated samples and perform baseband signal up and down conversion. The BBU then handles the computationally heavy baseband processing.

Softwarization of implementation combined with the physical decoupling of the BBU from the RRH introduces the possibility of virtualization and centralization. A single server may operate multiple BSs serving multiple cell sites. These sites only require RRHs and a communication link to the centralized server. By eliminating computational resources from access points, a more efficient architecture can be realised, in which processing resources are allocated from the pool based on demand. Enabling demand-based resource provisioning, in turn, allows for many low-cost, low-energy RRHs to be built to enhance coverage uniformity.

In the coverage area of a small cell, the number of active users can vary significantly. This, in turn, wastes energy with current peak-dimensioned designs due to a high probability of transmitters being idle [11]. Turning off idling base stations is one potential solution [12–14] but becomes problematic if one aims towards low-latency and high reliability communication. In such cases, when having a wake-up delay is unacceptable, energy reduction techniques constitute a preferable alternative.

Cellular networks exhibit spatial and temporal patterns in the load they experience [15, 16]. Daytime load is higher than during the night, when many users are asleep. Furthermore, due to diurnal variations between areas, load varies spatially by C-RAN instance as well as users move from residential areas to workplaces and back.

While RTOSs exist to solve latency and jitter challenges, they cannot be used in a virtualized environment since they cannot offer any guarantees when running on top of a non-real-time hypervisor controlling access to the actual hardware. Indirect access to hardware through the hypervisor further complicates management of timers and interrupts. Additionally, a real-time hypervisor's strict division of time between guests restricts opportunities for exploiting variable loads in guests.

Alternatively to RTOS, overprovisioning can be used to meet requirements. This method wastes both capital and energy. The predicted increase in the number of cell sites in 5G only compounds this problem.

In light of the above, it becomes necessary to investigate the ability of virtualizable, general purpose OSs and hardware to meet the timing requirements of BSs. In a non-real-time OS, packets can be lost due to missing the packet preparation deadline, denoted as late packets. In this work, soft-real-time is defined as a task that must meet its deadline on average but can tolerate occasional overruns [17, 18]. Conjointly, the relationship between energy efficiency and performance must also be investigated to determine design parameter trade-offs. Slowing down computing helps to run processors at more power efficient clock speeds.

By exploiting the diurnal variation, operators can save energy by reducing the processing power and thus energy

consumption during off-peak hours and in lightly loaded cells. This could be realised, for example, by grouping the processing of geographically close cells in the same C-RAN server. The power-performance profile of each server (and therefore cell group) can then be adjusted according its specific needs. Further refinement of the configuration granularity becomes possible on hardware support per-core adjustment of frequency and voltage.

Determining parameters with highest impact allows for the design of cloud systems better suited to the needs of virtualized BSs. The BS can reduce the amount of late packets by adjusting the radio link throughput and thereby the computational load. This opens up the possibility of leveraging a C-RAN platform to allocate computing resources intelligently. The following sections review research that has been conducted into latency-sensitive and energy-efficiency conscious software.

3. Related Work

3.1. Server Platforms for Time-Critical Systems. Management of system power consumption through online variation of processor voltage and frequency is known as dynamic voltage and frequency scaling (DVFS). Work presented in [19] proposes a statistical scheme to estimate and adapt to variation in load for latency-critical tasks. The authors consider variable arrival and processing times. They determine that queuing delays often dominate total latency. While similar in aim, the characteristics of the offered load in the present article are different. Due to the periodic nature of the LTE frame structure, arrival times are well determined and required completion times are known. Consequently, stale work items in queues can be discarded based on this known deadline. The BS also partially knows future load due to its control over resource allocations for connected devices it serves (see Section 2). The authors in [20] similarly focus on queuing. They however propose extending the time a task spends in the queue in order to obtain longer periods during which the processor resides in a low power state. Such a solution can be problematic in cellular systems, where communicating nodes must uphold strict timing in order to preserve synchronization. DVFS use in an embedded computer has been investigated in [21]. The authors studied the impact of voltage scaling on the worst-case execution cycle counts of their test programs. Unlike the present article, none of their test cases involved external interrupts caused by fronthaul data arriving at the network interface card (NIC) or other peripherals. Problems involved in operating low-latency systems involving substantial network communication are discussed in [22]. The authors of [22] discuss the general characteristics and broad outlines of potential solutions but no concrete solutions are offered. The aforementioned work demonstrates that power savings are possible but the case of interrupt-heavy periodic loads has received little attention.

3.2. Software-Based BS Implementations. Implementation of tightly latency constrained wireless communication systems on server hardware has been investigated before. In [1], the authors present their approach to running the LTE physical layer on Linux. In contrast to the approach used in this paper, they do not implement protocol layers above the physical. It is also unclear whether time-division duplexing (TDD) or frequency-division duplexing (FDD) is employed. In order to obtain satisfactory performance, the authors pin execution threads to specific CPU cores according to a hand-tuned pattern. In [23, 24], communication with the radio frontend occurred over a Peripheral Component Interconnect Express (PCIe) bus instead of an Ethernet connection as in the present work. Furthermore, in the latter case, the authors used the real-time framework Xenomai to introduce timing guarantees to the OS. Work presented in [25] reports on the methodology used to implement TD-SCDMA. The authors modified the Linux kernel's scheduling interrupt timer's frequency. This technique is no longer applicable to modern tickless versions of the kernel. Techniques applicable to real-time approaches, such as Xenomai, are more cumbersome since they attempt to enforce timing constraints on the entire BS software stack instead of only network packet processing. Interrupt patterns of IP-based traffic differ from those of PCIe. Studies into the performance of IP-based soft-real-time SDR are therefore needed.

3.3. Contribution. The above-presented related works considered either the problem of implementing latency-constrained communication systems or the reduction of energy consumption in processing local tasks. This article aims to combine both aspects and provide a scheme for reducing power usage while maintaining processing delays acceptably low. Energy savings are accomplished through CPU clock frequency scaling. Measurements quantify the dependence between the operating frequency and the BS's ability to complete tasks in a timely fashion. Factors taken into account include processing in the BBU, the radio frontend, and the network communication between them.

The present work explores the feasibility of implementing BSs on GPPs with a standard Linux OS kernel through tuning of hardware and OS parameters. The aim is to assess the suitability of a soft-real-time latency-critical cellular software to operate on cloud infrastructure. The techniques used do not require any modifications to the standard OS kernel or the hardware. This helps to ensure applicability in C-RAN execution environments.

4. Logical Functions in a Base Station

All BSs present the same functional interface to user equipment (UE) to allow interoperability. From the network's point-of-view however, they are not identical nor do they serve the same purpose. Some BSs, known as macro BSs, target a large coverage area called a macrocell. Complementing these are micro- and picocells serving demand in small hotspots. Large macrocells are suitable for serving fast moving users, since their large coverage area will result in less frequent handovers than if the user was served by more spatially limited access points.

While all the BSs are equipped with the same specification-mandated functions, they might be called at different rates and with different target parameters. As such, BSs can generate different computational loads. These differences provide opportunities for intelligent resource adaptation.

BS radio interface supports two-way communication. The BS receives data from the core network and prepares it for transmission over the air interface and conversely it takes the signal from the radio interface and converts it to bits to be transmitted to the core network. In order to compensate for channel errors, signals have to be acknowledged positively or negatively. The specification-mandated period [26] within which this must be accomplished sets a constraint on computing speed for decoding received data and sending out the acknowledgments.

The processing in BS can be split into physical layer processing and higher layer processing. The physical layer related processing is mainly execution of data flow type computationally heavy algorithms. Higher layers deal dominantly with users and protocol states. Figure 1 depicts the processing flow for data to and from the air interface. The BS's external connection to the mobile core network is beyond the scope of this work.

Functions dealing with digitized samples of the signal received at the antennas are collectively known as Layer 1 (L1). In 3GPP-specified cellular systems, L1 operates on groups of samples contiguous in time known as subframes or transmission time intervals (TTIs). In modern cellular systems, the TTI is the basic time unit for processing. The samples to be transmitted have to be prepared for each TTI and received samples have to be analyzed during certain TTIs.

The main functions in L1 are modulation, equalization, channel coding, and waveform processing. Processing load depends on the modulation and coding scheme (MCS) which the BS selects based on a mapping from the observed radio link quality. The MCS value selects operating parameters, such as constellation size and coding rate, for the modulator, demodulator, encoder, and decoder. In traditional systems, the choice of modulation accounts only for the quality of the channel between communicating nodes. In cloud-based systems, there exists an additional trade-off based on processing requirements. A higher MCS value allows for more bits to be transmitted but requires more processing for receiving those bits. The server can trade off lower data rates for reduced processing power requirements.

Above the physical layer reside the medium access control (MAC) functions. These primarily handle resource allocation, scheduling, and channel multiplexing. The MAC also carries the responsibility of generating acknowledgments and negative acknowledgments as well as responding to those received from the UE. In the LTE standard, the available response generation time is never shorter than four milliseconds. Resource allocation relies on these acknowledgments and on channel quality reports to inform the BS as to which transmission parameters are appropriate for the conditions experienced by each UE. Since the computational load is a function of user resource allocation, the scheduler can directly impact the BS's processing burden.

Connected to the core network are the BS functions dealing with IP packets. These provide header compression, encryption, integrity protection, and segmentation. Encryption and integrity protection provide security. Segmentation and concatenation functions are necessary to enable IP packets to fit the available radio resources. Header compression helps in this regard by eliminating needlessly redundant information from transmitted packets.

5. BS Implementation Using the ARF Platform

The agile radio framework (ARF) is a soft-real-time SDR platform designed to run on commodity computer hardware using a GNU/Linux OS as shown in Figure 2. In this work, it is used to implement an LTE BS in order to evaluate C-RAN latency and computation load issues. The ARF is designed to be compatible with a wide variety of different server platforms. As such, the ARF does not require special drivers, beyond those for the radio frontend in use, or modifications to the kernel of the OS. This entails that no hard-real-time extensions need be applied. While RTOSs provide tools to solve latency and jitter-related issues, they cannot do so when operating virtualized on top of a hypervisor. Additionally, compatibility of the ARF with standard software tools and libraries leads to wide portability. The same applies to online migration of virtual machines. Apart from the radio frontend, the ARF does not need any special hardware. However, use of such is not precluded by the design. For instance, one could use a graphics processing unit (GPU) to accelerate computations [27].

Architecturally, the ARF platform is divided into three main components as depicted by Figure 3. The lowest layer is the Virtual Hardware Enhancement Layer (VHEL) [28]. Its purpose is to serve as an abstraction layer for hardware nonidealities. Radio frontend hardware interfacing is done through the Universal Software Radio Peripheral Hardware Driver (UHD) [29]. Since the ARF does not assume hard-real-time guarantees from the OS, protocol processing may be late in the upper or lower layers. As explained in Section 2, cellular systems require transmitted and received samples to be ready for a specific TTI. In a non-real-time system, the processing could exceed its allotted time or be delayed past its deadline. In this paper, such events are termed lates. Upon occurrence of a late, the VHEL sends a preconfigured contingency subframe to the radio frontend to help maintain timing alignment. This could, for example, be a zero-filled subframe or a pregenerated subframe containing only pilots and synchronization signals if appropriate. In the receive direction, the VHEL will similarly mask lates and overflows by handing the protocol's physical layer a subframe containing the correct number of samples even if some were lost. The VHEL hides lates and therefore the soft-real-time nature of the platform from upper layers. These are written as if the samples would always arrive on time. Upper layers can therefore apply block-based processing while ignoring timing related issues.

The communication protocol's retransmission mechanisms can take care of performing a new transmission of data lost due to lateness of processing in the transmitter part

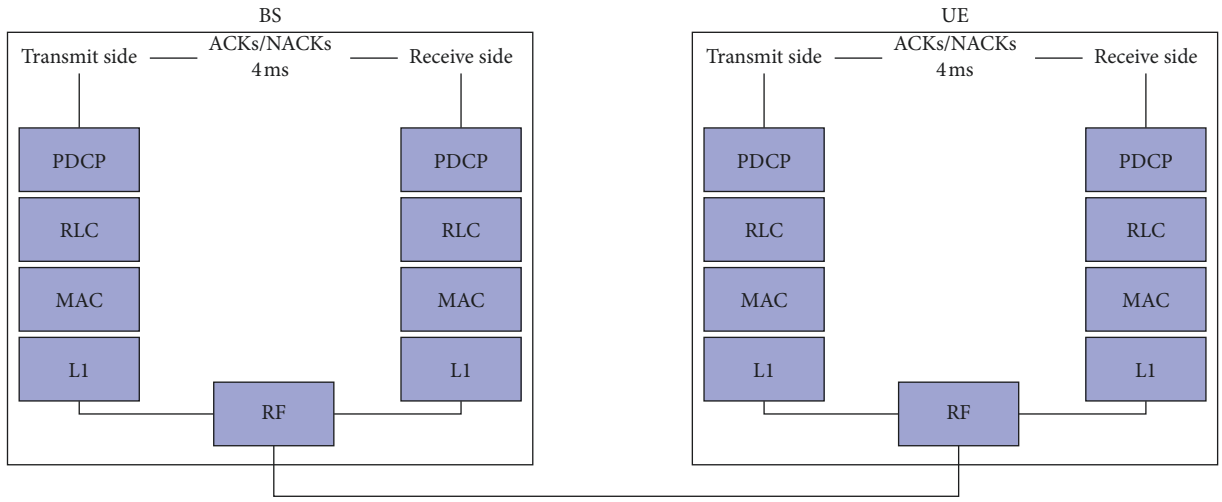

Figure 1: High-level overview of base station functions.

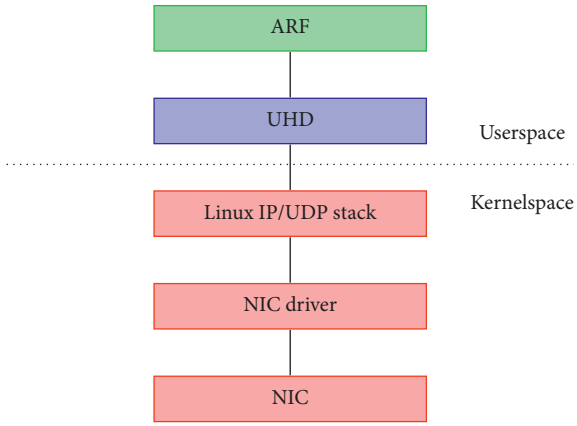

Figure 2: Overview of the main system components.

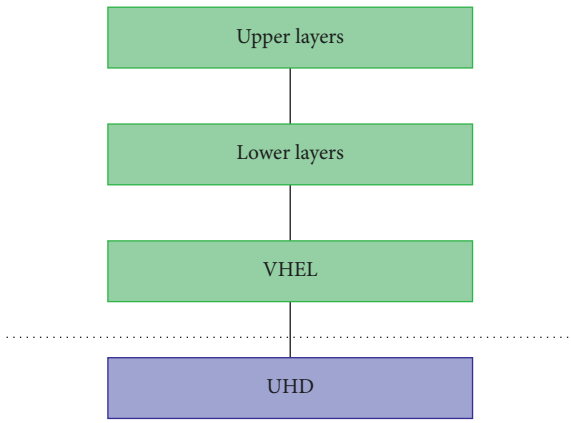

Figure 3: Layers in the ARF architecture.

of the ARF platform. From the communication protocol's point-of-view, this is no different than experiencing poor channel conditions, provided late processing occurs seldom enough. While this approach does not provide the same level of timing determinism as RTOS-based designs, it simplifies development. Code related to handling TTI timing and meeting deadlines resides only in the VHEL. Other code may be written as non-latency-critical software. In case performance proves insufficient, traditional software optimization techniques can be applied until deadlines can be met with some probability. Furthermore, an intelligent cloud platform could take advantage of the difference in processing load between different BSs as described in Section 2 to adapt its resources optimally.

6. C-RAN Processing Performance Model

The main challenge in C-RAN implementation is reduction of late packets. Whether a packet is late depends on the total packet processing time. This time is a function of the packet processing time within the BS stack T_{BS}, delays related to operating system functionality T_{OS}, and delays related to data transmission over the fronthaul link T_{f}. Knowledge of the latter is required in order to estimate how far ahead of the deadline a packet must be sent to arrive in time at the frontend.

Accurate total processing time information might not be available to the C-RAN. In practical systems, it is easier to measure the number of late packets as a function of these various delays $F_{\mathrm{P}}(T_{\mathrm{BS}}, T_{\mathrm{OS}}, T_{\mathrm{f}})$. The function F_{P} provides a tool for adapting BS processing. To achieve a given packet outage target P_{out}, the C-RAN should manage delays such that

$$F_{\mathrm{P}}(T_{\mathrm{BS}}, T_{\mathrm{OS}}, T_{\mathrm{f}}) \leq P_{\mathrm{out}}. \tag{1}$$

While allocating its resources, the C-RAN can track F_{P} in real time. Optimal allocation raises the question of whether F_{P} captures the system's characteristics sufficiently well and which parameters are the most significant for system performance. Function F_{P} is implementation dependent. However, system-specific dependencies can be measured and learned for each particular platform at run time.

The ARF platform is used as a measurement tool for identifying cloud processing bottlenecks while executing an SDR BS load. Execution time is subdivided into two parts: cellular BS-related functionality and operating system housekeeping activities. The former constitutes the lion's share of the load. In turn, it is composed to two components. The first is air interface processing, comprised of relatively computationally heavy physical layer algorithms executed each in TTI (1 ms interval in LTE). Higher layer communication protocol processing executes separately from the physical layer and presents a comparatively much lighter processing burden. The second component includes all noncore-functionality processing, termed management processing, such as data copying, OS function calls, and fronthaul data transmission. Contention for these resource as well as OS kernel scheduling decision can influence whether processing completes on time. Overall C-RAN behavior is characterized by how the system management and BS processing work together. This was investigated through the insertion of measurement points into the BS and management parts of the ARF code.

The results presented in this article were obtained by running a partial LTE Release 8 payload in TDD mode on the ARF. Consequently, the TTI duration is 1 ms. The measurements were done while sending data over the air. Performance of the TX processing impacts the RX processing and vice versa. Processing-time overruns in one will reduce the CPU time available for the other. This stems from the need to know whether to send a positive or negative acknowledgment and, on the transmit side, whether the retransmission buffer can be cleared after a successful reception by the remote node. UEs must first receive grants from the BS before they know when and with what parameters they may transmit. This limits the available time to prepare data to the time between receipt of the grant and the transmission time.

Packet processing time (including transmission to RRH) should be less than the TTI duration. Each component of the packet late function F_P contributes to overall processing time differently. The delay T_{BS}, related to user load, is a linear function of the computational demands on the CPU. The more data the UEs want to receive, the more work the BS must perform to transmit it to them. In order to fit a greater quantity of data into the same, fixed, amount of spectrum, more complex modulation, equalization, and channel coding schemes must be employed. Doing so increases computational complexity. Increased load can be managed by adding more processing capacity. This could, for example, be more or faster CPUs as well as accelerators.

Fronthaul transmission-related delays are composed of data transmission and endpoint processing. For dedicated fronthaul links, the data transmission delay T_f is largely fixed and adds latency without any means for the BBU to compensate. Endpoint processing, on the other hand, can be sped up in the BBU. Typically, it is the OS that handles network-related tasks. It is therefore possible to obtain improved performance by switching to more capable network hardware, a newer kernel, or different OS.

The main contributor to OS-related delays T_{OS} is interrupt processing. It causes delay and jitter due to context switches to interrupt service routines and handlers [35]. Context switches degrade performance even if they are not computationally expensive as they might cause data to be evicted from cache, new processor state information to be loaded, and virtual memory translation lookaside buffers to be flushed [36]. Since context switches can be caused by aperiodic background processes and hardware interrupts, their effect on an SDR platform—or any other software—is not easily predicted or quantified by said software itself. Mitigation strategies must therefore extend beyond the SDR code base into the operating system, hardware, and load management.

7. Experimental Setup

Measurements to quantify the contribution of each factor of F_P (equation (1)) were carried out using two different systems. These measurements serve to quantify the impact of general purpose hardware and operating systems on cellular BS implementation performance. Table 1 presents the test systems' main components while Table 2 lists the Linux kernel boot arguments. Both systems were directly connected to their respective frontends using Gigabit Ethernet. The ARF SDR platform was configured as an LTE Release 8 BS with one transmitter and one receiver antenna. Assessing the impact of protocol processing was done through the use of the data plane. Turbo Code decoding was used to create a uniform and constant load. Five iterations of the decoder were applied to 25 resource blocks (RB) in the LTE subframe structure. Modulation and coding scheme (MCS) 9 was used as defined in LTE Release 8 for uplink data transmissions.

The operating system used was Ubuntu 16.04 LTS. In an effort to reduce interrupt and CPU contention for the BS code, OS boot-time parameters were modified from their defaults. The aim of the changes is to improve the performance of the ARF platform by dedicating resources. Listing 1 gives the parameters applied to the first system's Linux kernel on boot and Listing 2 presents the same for the second system. The "intel_pstate = disable" parameter was used to prevent dynamic control of the operating frequency in order to enable consistent measurement runs. The OS's scheduler was instructed to avoid CPU cores 2–7 and 10–15. Doing so reserves them for use by the BS code. The parameters in Listing 1 and Listing 2 were in addition to the "root=" parameter as well as the default values of Ubuntu 16.04 for each system, given in Listings 3 and 4, respectively. No additional kernel boot settings were applied to System 3.

In addition to boot-time parameters, measures were taken after system start-up to further improve performance on System 1 and System 2. The governor for all processor cores was set to the "performance" setting. The energy-performance bias parameter was also changed. It was set to indicate to the CPU to prefer higher performance over lower energy consumption. This was done in an effort to minimize latency and increase the repeatability of the data since the decisions of the CPU are opaque to the user. The SDR platform process was also executed with a high priority. This

Table 1: Computer system configuration.

Component	System 1	System 2	System 3
CPU	Intel Xeon D-1541 [30]	Intel Xeon E3-1230 v3 [31]	Intel Xeon E5-1650 v4 [32]
RAM	32 GB 2133 MHz DDR4	16 GB 1600 MHz DDR3	32 GB 2400 MHz DDR4
NIC	Intel I350	Intel 82574L	Mellanox Connect4-X LX Intel XL710
Frontend	NI USRP N200 [33]	NI USRP N200 [33]	NI USRP X300 [34]

Table 2: Kernel configurations for systems 1 and 2.

Listing 1	intel_pstate = disable isolcpus = 2–7, 10–15 rcu_nocbs = 2–7, 10–15
Listing 2	intel_pstate = disable
Listing 3	ro quiet splash
Listing 4	ro quiet splash vt.handoff = 7

helps to ensure that the OS's scheduler tries to minimize interruptions caused by other tasks requesting processor time. Ensuring constant operating parameters reduces jitter from transitions between them and helps avoid unnecessary delays. Such transitions were observed to occur frequently. One possible reason for this is that relatively low CPU utilization rate led the OS to reduce clock frequency in order to save power as it believed the system to be relatively unloaded. In fact, the system was likely busy moving data or waiting for it. Waking up from a low power state when the data does arrive takes more time than resuming processing from a fully on state. The cset application [37] was used to set the ARF code to run on the isolated CPUs. On System 1, two cores and their adjoining Hyperthread cores were kept for other processes while on System 2, the corresponding number was one. On System 3, the CPU governor was set to "performance" and the interrupt coalescing value to 10 μs.

Although the measurements presented in this work were obtained on a physical host without virtualization, we argue that the results provide meaningful insights into C-RAN implementation for three reasons. Firstly, a C-RAN differs from a public cloud service in that the platform is purpose-built to host cellular BBUs under the control of the same administrative entity, i.e., the network operator. As such, fewer abstraction layers are required than in a public cloud. For the same reasons, direct access to hardware resources can be granted to guests. Furthermore, hardware-assisted virtualization technology, such as single root input/output virtualization (SR-IOV), enables a single device to be shared amongst the guests at a hardware level with low overhead [38]. Secondly, being a purpose-build platform also makes it feasible to provide an interface (i.e., paravirtualization) for the guests to communicate their load status to the hypervisor. The latter can then utilize the similarity of the diurnal cycle induced load (see Section 2) across neighboring BBUs to scale down the CPU frequency of the whole system. Thirdly, an operator's business model differs from a public cloud provider. The latter might oversubscribe resources to increase revenue [39]. Another difference is that public clouds do not know a priori what load type their customers will run. Infrastructure must therefore be designed to be generic. A C-RAN platform, on the other hand, knows the load type to be executed and places meeting latency requirements first. Consequently, resource allocations can be tailored to the task at hand. Furthermore, since payloads in a C-RAN platform originate from trusted users, less need for hardware abstractions and security isolation overhead exists. Performance, therefore, behaves more akin to a physical host, yet still provides the benefits of consolidation by enabling multiple BSs to be hosted on a single server.

Measurements were performed to determine the impact of two different system parameters on the performance of the ARF. The first one was the clock frequency of the CPU. It was varied from highest to lowest supported by the CPU (see Table 3). CPU clock frequency measurements were done to study the feasibility of reducing power consumption without compromising performance. A lower clock frequency enables operating voltage to be dropped, which in turn leads to power savings. The CPU's operating speed impacts the T_{BS} parameter of F_{P}. The second type of measurement involved varying the interrupt coalescing behavior of the NIC (see Section 3.3). Table 3 shows the parameters used for interrupt coalescing. The value labeled "rx-usecs" indicates the amount of time in microseconds that the NIC was configured to wait for further network packets to arrive before notifying the CPU in an effort to improve the interrupt-to-payload ration. Interrupt coalescing was selected as test parameter to assess the impact of OS processing (T_{OS} in F_{P}) on BS performance. A greater number of NIC interrupts result in more processing time spent in kernel space. This does not significantly change the computational load as each network packet still needs to be processed but does reduce the number of context switches required.

8. Results and Analysis

Results from the measurements are reported as the probability of the subframe processing being late. It is computed by normalizing the number of late subframes by the total number of subframes. This metric was chosen instead of the more conventional CPU usage as it better reflects the objective: to process received subframes and prepare new ones to transmit within the alloted time. It would indeed be possible for the processor to be relatively unoccupied but to have data be delayed by other factors (for example, network stack processing or interrupt moderation by NICs). CPU clock frequency and interrupt coalescing measurements

Table 3: CPU frequency and interrupt coalescing test parameters.

Test name	Duration per value	Starting value	End value	Step size
rx-usecs (μs)	30 min	10	300	10
cpu-freq (MHz) system 1	15 min	1000	2100	100
cpu-freq (MHz) system 2	15 min	1000	3300	100

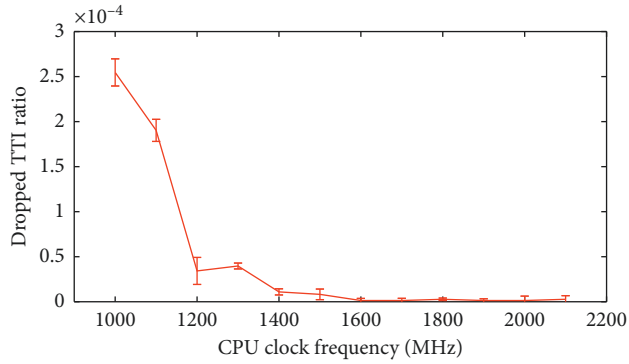

Figure 4: System 1: dropped TTI ratio for each clock frequency value recorded. The red line is the median with the vertical bars depicting ± one standard deviation. 5 MHz bandwidth.

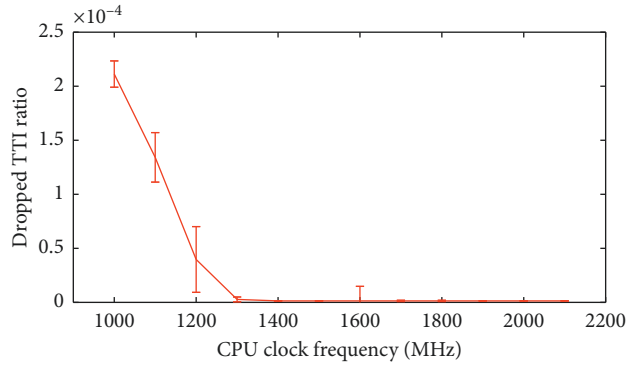

Figure 5: System 1: dropped TTI ratio for each clock frequency value recorded. The red line is the median with the vertical bars depicting ± one standard deviation. 10 MHz bandwidth.

were carried out using System 1 and System 2 while the NIC offload measurements were performed using System 3.

8.1. Performance per MHz. Comparing Figures 4 and 5, performance can be seen improving from the lighter processing load to the more demanding one. Since the ARF platform can handle higher data rate (load) better, improved performance at a higher load suggests computational capacity is not the sole factor influencing performance. To confirm that delays outside of the SDR platform impact performance, measurements were carried out to determine the effect of additional system delay in processing incoming network traffic. The NIC's interrupt coalescing timeout was varied to introduce extra delay. Despite load, bandwidth, and CPU clock frequency remaining constant, Figure 8 shows an increase in the ratio of dropped TTIs.

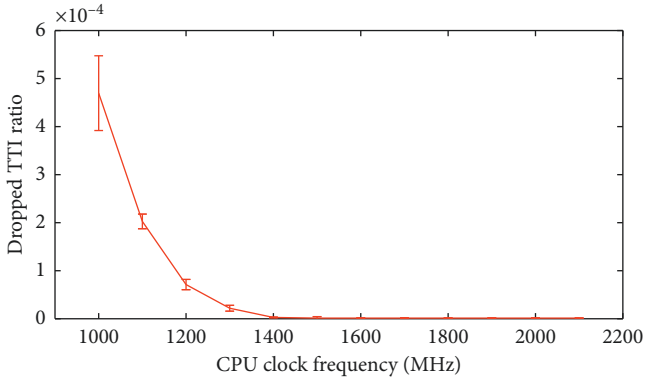

Figure 6: System 1: dropped TTI ratio for each clock frequency value recorded. The red line is the median with the vertical bars depicting ± one standard deviation. 10 MHz bandwidth with 25 RB load.

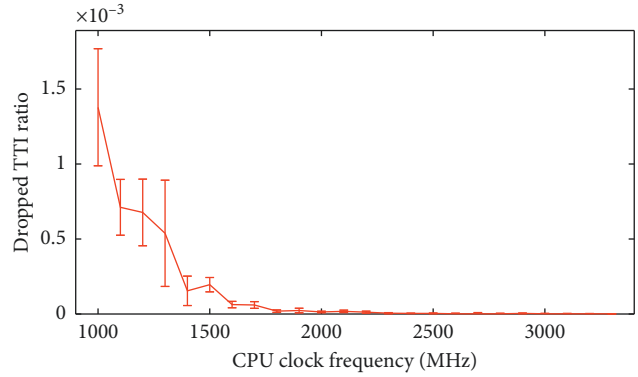

Figure 7: System 2: dropped TTI ratio for each clock frequency value recorded. The red line is the median with the vertical bars depicting ± one standard deviation. 10 MHz bandwidth with 25 RB load.

Even though a subframe may be ready in time in the BBU, it still needs to be transported to the RRH. When setting the frequency of the CPU, frontend transmission delays must also be taken into account. For receiver-side data processing, the same principle applies in reverse; not all of the TTI duration is available for baseband processing. This reduction in available processing time must be taken into account in the design of the SDR platform and the configuration of the OS it runs on. Unlike RTOS running on ASIC-based designs, a margin should also be included to protect against jitter in network processing caused by competing processes running on the same machine.

The measurement with heavy protocol processing load leads to the situation depicted in Figure 6. The shape of the curve remains the same but the ratio of dropped TTIs below the threshold frequency rises. Performance above this threshold point remains essentially unchanged. Operating the CPU at full frequency therefore needlessly increases energy expenditure without providing a corresponding improvement in the timeliness of computations or the performance of the platform in general. It is already operating at effectively the same performance as a dedicated design with preset timing and scheduling.

A potential alternative would be to execute the job as quickly as possible using the highest performance setting and then enter a low-power state. Such an approach may not be suitable in all cases. Resuming normal execution from a low-power state incurs a delay and consumes full power [40]. Therefore, in order to be beneficial, the length of the idle period must be long enough that the energy and performance penalty [41] of transition into a power saving state is recouped. The latency is of particular concern in cellular systems. Processing a subframe requires obtaining all its constituent samples from the radio frontend. These do not arrive at the same time in one transaction but are instead sent in chunks, the size of which is determined by the Ethernet maximum transmission unit (MTU) size. Frames are sent as soon as they are filled with samples. This means the C-RAN platform must continuously process incoming data from the network and thus requiring CPU involvement. Delaying processing of arriving data means accumulating work to be done once the last sample is received. This backlog of work then creates extra delay in the handling of the subframe as pending data frames must be handled first. Similarly, if TTIs get shorter or subframes are processed in smaller parts to reduce latency, the intervals available for power states decrease even further. However, the frequency-scaling method presented in this work is not mutually exclusive with other power management techniques but rather a complementary approach. For example, it may be possible to hand over users from several lightly loaded BBUs to one. Then, the all but one could be put into power saving mode while the active one executes with the lowest power state providing the required performance. How to combine these techniques as part of an overall solution is left up to future work.

Figure 7 indicates that the existence of the energy-saving frequency threshold is not specific to System 1. System 2 exhibits the same type of behavior but with a greater potential energy saving due to a higher maximum CPU operating frequency.

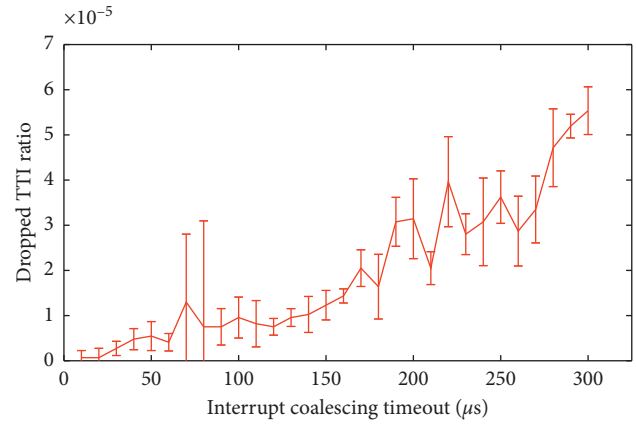

Figure 8: System 1: dropped TTI ratio for each interrupt coalescing timeout tested. The red line is the median with the vertical bars depicting ± one standard deviation. 5 MHz bandwidth.

Table 4: Initial value multipliers used in generating initial values for least squares curve fitting.

Multipliers	−1000	−100	−10	1	10	100	1000

Table 5: Smallest MSE value for tested model coefficient counts 1–9 for f_0 for System 1.

	1	2	3	4	5	6	7	8	9
c_1	—	—	—	—	−6.6338e + 08	—	—	3.9947e + 06	−4.3060e + 05
c_2	—	—	—	—	—	2.0958e + 10	1.2865e + 05	−3.6605e + 03	—
c_3	—	—	—	—	—	−1.9060e + 07	−119.1427	—	0.3559
c_4	—	—	—8.6528e + 09	—	—	—	—	−2.3198e + 03	—
c_5	—	—	—	—	—	—	—	—	0.0644
c_6	—	—	—	−43.3500	—	—	—	—	—
c_7	—	—	—	−2.5526e + 04	—	1.0976e + 13	9.3501e + 07	9.9289e + 05	—
c_8	—	—	—	—	−6.4473e + 05	—	—	−930.6551	−822.0543
c_9	—	—	—	—	—	−9.0785e + 06	−83.9962	—	0.7473
c_{10}	—	—	—	—	−4.9180e + 06	1.1812e + 07	—	—	—
c_{11}	—	—	—	—	—	—	—	—	0.0669
c_{12}	−2.0083	−2.0203	−8.6256e + 06	—	—	—	—	—	—
c_{13}	—	—	—	—	−33.3167	−18.6360	−16.2634	12.6242	16.5625
c_{14}	—	—	0.0066	0.0019	0.0333	—	—	−0.0175	—
c_{15}	—	—	—	−1.8772e − 06	—	—	1.3552e − 05	4.8688e − 06	—
c_{16}	—	—	—	—	—	—	—	—	0.7894
c_{17}	—	—	—	—	—	—	—	—	—
c_{18}	—	6.4146e − 05	—	—	—	—	2.1030e − 05	—	−4.9713e − 04
MSE	0.7722	0.1178	0.0028	0.0020	0.0015	8.9748e − 04	7.2662e − 04	6.9194e − 04	6.6601e − 04

8.2. Performance Modeling. C-RAN server run-time adaptation requires learning the BS's performance vis-à-vis its load and to be able to predict the impact of changes in operating parameters. One way to accomplish this is to extract a dependency function, the parameters of which can be learned in real-time. Such a function was sought using collected performance data. Analysis of the data (see Figures 4–7) suggested that the impact of CPU frequency can be divided into two distinct regimes: one in which it majorly affects the late rate and one in which it does not. Due to this shape, two function families were selected:

$$f_0(c,x) = \left(\frac{g_1(c,x)}{g_2(c,x)}\right)^{g_3(c,x)},$$
$$f_1(c,x) = g_1(c,x) * \exp\left(\frac{g_2(c,x)}{g_3(c,x)}\right). \quad (2)$$

The criteria for the selection of the equations' form were being a function of CPU frequency and sampling rate, possessing two regions (steep and shallow), and having a suitably rapid transition from one region to the other. Parameterwise, CPU frequency was selected since it is user tunable and significantly affects both performance and energy consumption. Sampling rate similarly heavily impacts the amount of data needed to be processed. Through preliminary analysis of early measurements, these two factors were picked as being the most significant.

Studying the effect of sampling rate provides one of the defining characteristics of C-RAN, namely, a steady and constant inflow of samples to process. The analysis could, however, be made more general by considering a generalized load factor. Such a load factor would be constituted of the traffic-independent part (i.e., sampling rate) and the traffic-dependent part (i.e., per-resource block processing). The latter would be subject to the diurnal patterns discussed in Section 2. The analysis of such a generalized model is left up to future work.

While the general form of the function could be determined by inspection of the collected data plots, parameter combinations are too numerous for manual determination to be practicable. Instead, an exhaustive search through all the parameter combinations determines the best selection, as well as their coefficients as presented in [42]. Functions g_1, g_2, and g_3 (equations (4)–(6)) contain these coefficients:

$$g_1(c,x) = c_1 + c_2 f + c_3 f^2 + c_4 R_s + c_5 R_s^2 + c_6 f R_s, \quad (3)$$

$$g_2(c,x) = c_7 + c_8 f + c_9 f^2 + c_{10} R_s + c_{11} R_s^2 + c_{12} f R_s, \quad (4)$$

$$g_3(c,x) = c_{13} + c_{14} f + c_{15} f^2 + c_{16} R_s + c_{17} R_s^2 + c_{18} f R_s. \quad (5)$$

Fitting of the model was done using the least squares method in MATLAB [43]. Mean-squared error (MSE) was used as the goodness-of-fit metric. The least squares method exhibits sensitivity to its starting point. Multiple initial values were therefore employed. Initial values were obtained as two-step process. First, random values were obtained by multiplying a random number uniformly distributed in $[0, 1]$ with a multiplier. Table 4 lists the multipliers used. Each parameter of the model received an independently selected value. An initial fitting was then conducted. A subset of the lowest MSE coefficients was retained to serve as initial values for subsequent fittings.

Iterations of the search potentially yield unsuitable results. This may occur due to ill-conditioned Jacobians or rank deficiency. Any set of coefficients producing issues of the aforementioned type, as reported by MATLAB, was rejected. Additionally, significance of each parameter to the overall model formed an additional criterion. Parameters more likely than $\alpha = 0.05$ to result from the null hypothesis resulted in rejection of the generated model.

Table 6: Smallest MSE value for tested model coefficient counts 10–18 for f_0 for System 1.

	10	11	12	13	14	15	16	17	18
c_1	3.7982e+05	—	2.8737e+05	4.5276e+04	2.9898e+05	−8.6570e+04	−7.8668e+04	−4.0430e+04	−4.0439e+04
c_2	−831.8001	7.6956e+04	−470.6707	−80.3875	−486.4683	−296.8135	−314.7858	−339.6572	−339.6654
c_3	0.4543	−69.6829	0.1923	0.0352	0.1973	0.3688	0.3633	0.3606	0.3606
c_4	−818.7168	−1.6219e+06	77.4133	92.0675	—	−626.3216	254.2513	−699.3099	−696.2262
c_5	—	—	—	—	—	—	−52.4260	−40.5088	−40.5092
c_6	—	1.4257e+03	—	—	0.0828	−0.9352	—	0.3189	0.3159
c_7	—	5.8681e+07	2.6369e+05	4.5355e+05	2.5348e+05	−2.2821e+05	−2.9090e+05	−2.3938e+05	−2.3938e+05
c_8	−352.6329	—	−438.8796	−885.2676	−428.2140	−115.1857	−42.5138	−82.4825	−82.4819
c_9	0.3336	−51.4092	0.1842	0.4318	0.1846	0.3288	0.3033	0.3024	0.3024
c_{10}	−469.5781	−2.0383e+06	75.4453	91.7248	357.4965	—	1.4011e+03	0.0893	0.1160
c_{11}	23.2165	—	—	—	—	0.2195	−52.3259	−40.3876	−40.3865
c_{12}	—	1.8421e+03	—	—	−0.2753	−1.5678	−1.1497	−0.3839	−0.3839
c_{13}	—	−12.4182	128.0416	−5.5717	73.7797	−73.4409	−45.1022	−48.4219	−48.4182
c_{14}	−0.0117	—	−0.2174	0.0062	−0.1251	0.0696	0.0411	0.0442	0.0442
c_{15}	1.1659e−05	1.0135e−05	9.1904e−05	−1.1883e−06	5.2680e−05	−2.1141e−06	—	−6.7999e−14	−6.7999e−14
c_{16}	—	—	0.1030	0.1273	−4.9831e−05	—	−0.1540	0.0792	0.0795
c_{17}	—	—	—	—	—	−0.0037	−0.0034	−0.0036	−0.0036
c_{18}	—	2.6237e−05	—	−7.8043e−05	7.4276e−05	6.1047e−05	2.1484e−04	—	1.7027e−07
MSE	6.5194e−04	6.4732e−04	5.1029e−04	5.8179e−04	4.5423e−04	6.1349e−04	6.2157e−04	6.3755e−04	6.6866e−04

Table 7: Smallest MSE value for each tested model coefficient counts 1–9 for f_1 for System 1.

	1	2	3	4	5	6	7	8	9
c_1	—	0.1728	—	—	0.0157	—	$-5.3419e+05$	-0.0084	-0.0036
c_2	—	—	—	0.0366	—	-0.1128	—	$8.4137e-06$	$3.6577e-06$
c_3	—	—	0.0079	—	—	$1.1375e-04$	0.5342	—	—
c_4	—	—	—	—	—	—	—	—	—
c_5	—	—	—	—	—	—	—	—	—
c_6	—	—	—	—	—	—	—	—	—
c_7	—	—	—	—	—	$-9.1552e+05$	—	$-6.5538e+05$	—
c_8	—	—	5.5567	—	25.7958	915.3064	$1.5507e+03$	436.4609	355.4889
c_9	—	—	—	0.8146	0.8146	—	-1.5507	—	—
c_{10}	—	—	—	—	—	—	—	$-4.4712e+04$	—
c_{11}	—	—	—	—	—	—	—	—	—
c_{12}	—	—	—	—	—	—	—	—	12.1912
c_{13}	—	—	-620.3365	$-2.3393e+05$	$1.7225e+05$	—	$-5.0350e+04$	$-2.1597e+04$	$2.0170e+05$
c_{14}	—	—	—	—	-331.6080	—	—	—	-476.7543
c_{15}	—	—	—	—	0.1656	-0.0266	0.0536	—	0.3077
c_{16}	—	—	—	—	—	203.1686	-29.6142	$2.4683e+04$	$-8.7337e+03$
c_{17}	—	—	—	—	—	—	—	—	—
c_{18}	$6.4146e-05$	$6.4146e-05$	—	0.4405	—	—	—	-29.0912	9.8531
MSE	0.1178	0.0898	0.0030	0.0025	0.0016	$8.4944e-04$	$7.3278e-04$	$6.6535e-04$	$6.4281e-04$

Some parameter combinations yielded results with imaginary components. These were also rejected.

Tables 5 and 6 present the lowest MSE score for each parameter count when fitting data recorded on System 1 using f_0. The corresponding values for f_1 are given in Tables 7 and 8. Similar results for System 2 are omitted for brevity. They followed a similar pattern to the results for System 1.

After an initial sharp drop in MSE between one and six parameters, performance remains mostly stagnant from seven parameters upwards. For f_0, the lowest MSE occurs for 14 parameters and for f_1, at 13. Lower parameter count models are preferable as they are less likely to overfit, which degrades predictive accuracy, and since they are less costly to fit parameters for processingwise. Therefore, we select f_0 at seven parameters as the most suitable model (its MSE being lower than f_1 at the same number of parameters). Equation (6) presents the selected model:

$$f_{\mathrm{m}}(f) = \left(\frac{1.2865e+05 * f - 119.1427 * f^2}{9.3501e+07 - 83.9962 * f^2}\right)^{-16.2634+1.3552e-05*f^2+2.1030e-05*f*R_s}. \quad (6)$$

Figure 9 illustrates the selected model's fit for System 1 data. To assess the impact of specific system characteristics on performance, Figure 10 presents the fit f_{m} with the data recorded for System 2. It can be observed that while the general shape of the curve remains similar, there is more divergence between modelled and recorded than in Figure 9. Residual plots (Figures 12 and 13) for the selected model indicate a different pattern on both systems. The quality of the fit improves when the model is retrained based on System 2 data as shown in Figure 11. This indicates that the general form of the function applies to multiple platforms. Machine-specific variations are expected since hardware, OS, driver, and load differences impact behavior. Completely accounting for all such variations in practice presents an almost infeasible task.

Determining when diminishing returns set in as a function of f_{m} accomplished by analysing the relationship between the dropped TTI ratio and the CPU clock rate. Beyond the threshold frequency, performance stops increasing. A practical SDR platform implementation may track this information and use it to build time averaged statistics. These in turn can be utilized in online computations of the predictive equation's parameter values reflecting the current state of the execution environment. Shorter averaging times lead to a more responsive estimation of the required CPU frequency while longer ones reject transients better.

In addition to the statistics collection above, SDR systems may also utilize communication protocol scheduler data for performance-energy optimizations. Since in an LTE system UEs may only transmit when given a grant by the BS, the latter knows to a large degree its receiver-side processing load some period of time into the future. It is not possible to account for all incoming traffic (e.g., random access bursts), so a margin should be left to avoid underestimating the required computational resources. No fluctuation margin was included in the values computed in this section as that is a system design issue and therefore dependent on the use-case.

8.3. Network Processing Impact. Cloud infrastructure exhibits not only a non-real-time computational

Table 8: Smallest MSE value for each tested model coefficient counts 10–18 for f_1 for System 1.

	10	11	12	13	14	15	16	17	18
c_1	—	—	0.1741	0.1437	−11.4554	0.4718	—	$-6.8403e+04$	$-6.8400e+04$
c_2	—	—	$-2.9720e-04$	$-2.4694e-04$	—	$-4.9200e-04$	−576.9956	−538.8250	−538.8220
c_3	—	$9.1837e-12$	$1.2587e-07$	$1.0493e-07$	$1.1709e-05$	$1.2706e-07$	0.5772	0.6074	0.6074
c_4	$-1.7123e-04$	$-3.5257e-04$	—	$2.8794e-04$	−6.0084	0.0020	$1.2289e+04$	$1.2849e+04$	$1.2849e+04$
c_5	—	—	$4.2382e-05$	—	—	—	−14.3642	−17.3039	−17.2991
c_6	$1.7197e-07$	$3.5343e-07$	—	—	0.0060	—	−11.9380	−12.4240	−12.4239
c_7	$-2.6428e+05$	−355.3216	—	—	$2.5453e+07$	$1.0374e+06$	0.5739	0.5739	0.5739
c_8	—	—	$-4.4051e+03$	$1.7857e+03$	$-2.4494e+04$	151.6592	−569.8907	−575.8266	−575.4624
c_9	0.0012	—	2.8574	−1.1618	—	—	0.6176	0.6249	0.6246
c_{10}	—	—	—	—	—	$4.0681e+05$	$1.6873e+04$	$1.6622e+04$	$1.6628e+04$
c_{11}	−39.7750	−0.1177	—	375.9358	909.4912	88.7467	−89.1419	−90.8673	−90.9799
c_{12}	−1.4762	—	7.1416	—	−52.2593	−373.5502	−16.6696	—16.4146	−16.4177
c_{13}	$4.0817e+04$	−29.1972	$-3.7072e+06$	$1.5405e+06$	$-1.3173e+06$	$2.5026e+07$	—	—	−145.8319
c_{14}	−60.6124	0.0640	$5.6500e+03$	$-2.3883e+03$	$2.0404e+03$	$-4.6417e+04$	22.6284	22.8185	23.0005
c_{15}	—	$-6.5701e-05$	−2.1828	0.9488	0.0285	21.9223	−0.0295	−0.0297	−0.0298
c_{16}	$4.1613e+03$	7.8850	$5.1312e+04$	$-1.6177e+04$	$-4.1067e+04$	$8.3393e+04$	−580.9436	−565.7905	−564.1394
c_{17}	—	−0.0100	—	70.0923	450.5005	173.7471	7.1425	7.0927	7.0987
c_{18}	−4.5610	−0.0080	−57.1656	17.6446	15.6408	−64.2595	0.6673	0.6528	0.6513
MSE	$6.1681e-04$	$6.2235e-04$	$5.0068e-04$	$4.8199e-04$	$6.2180e-04$	$6.2916e-04$	$6.0026e-04$	$6.1458e-04$	$6.1423e-04$

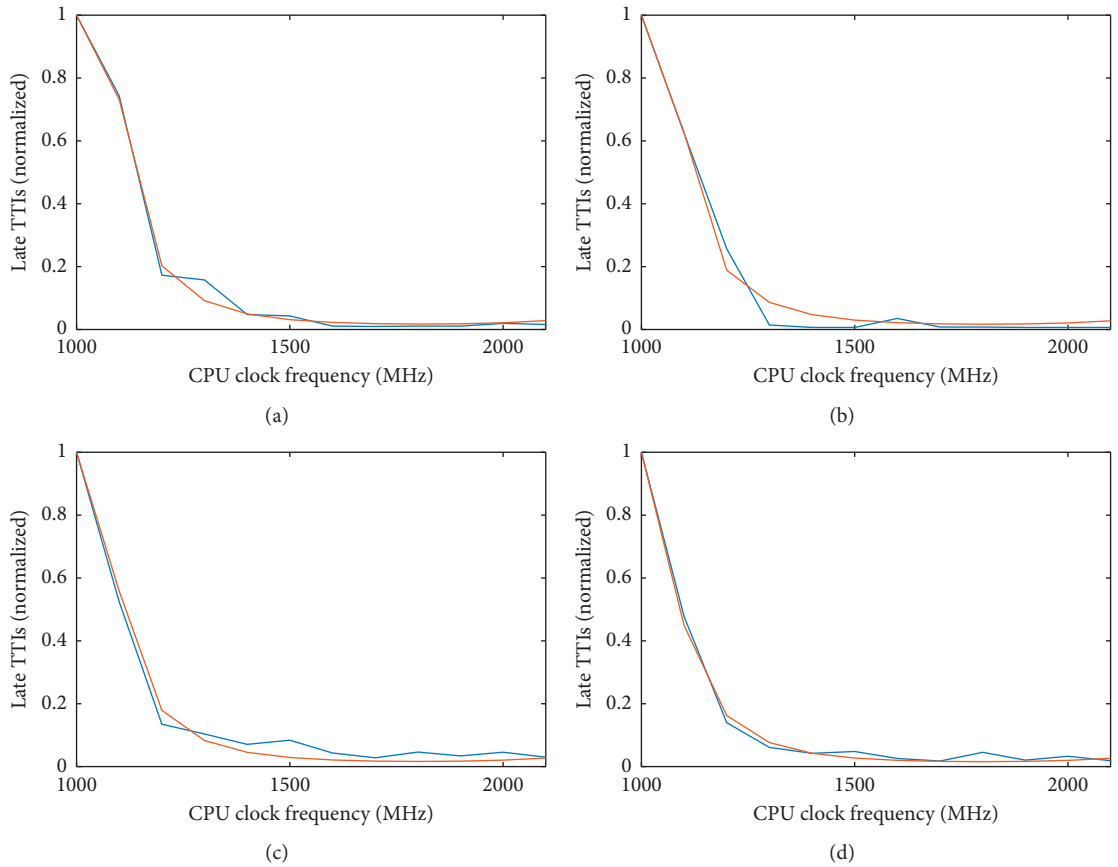

Figure 9: System 1: comparison of actual and predicted lates for f_m. (a) 5 MHz bandwidth. (b) 10 MHz bandwidth. (c) 15 MHz bandwidth. (d) 20 MHz bandwidth.

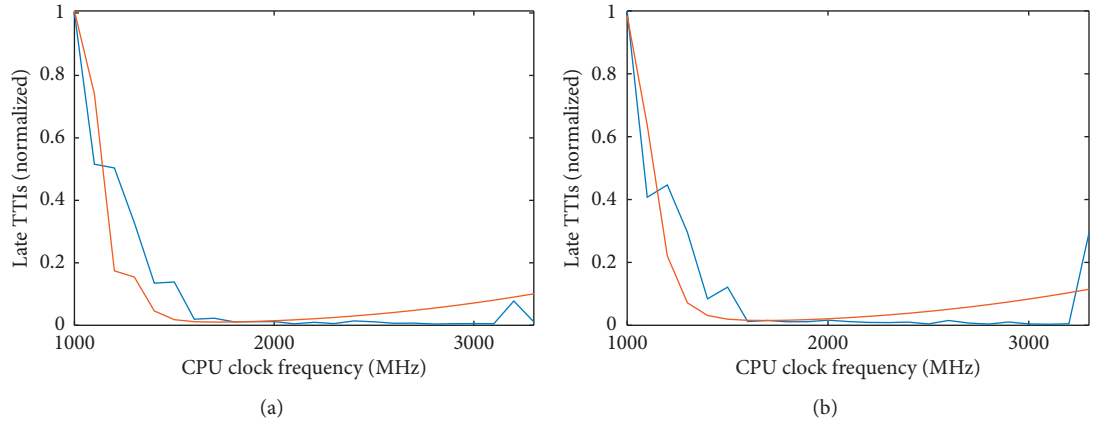

Figure 10: Continued.

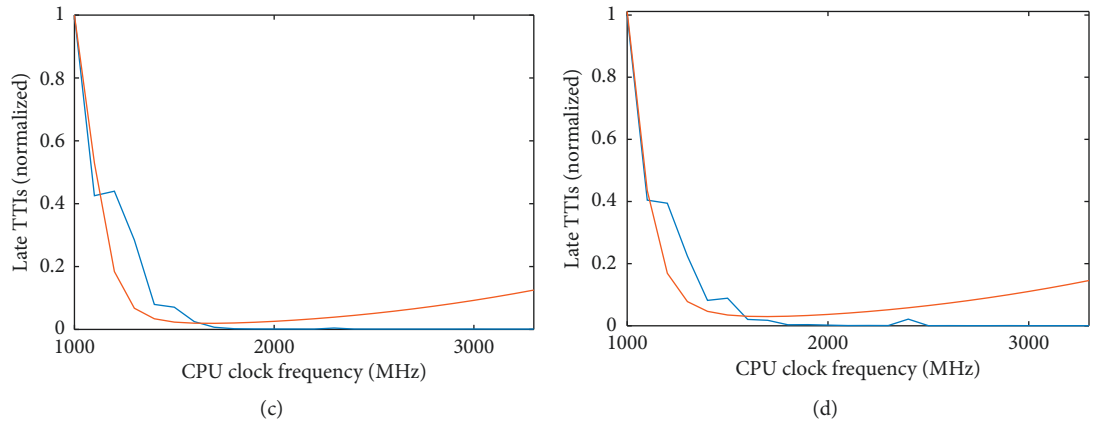

FIGURE 10: System 2: comparison of actual and predicted lates for f_m using the coefficients computed for System 1 f_m. (a) 5 MHz bandwidth. (b) 10 MHz bandwidth. (c) 15 MHz bandwidth. (d) 20 MHz bandwidth.

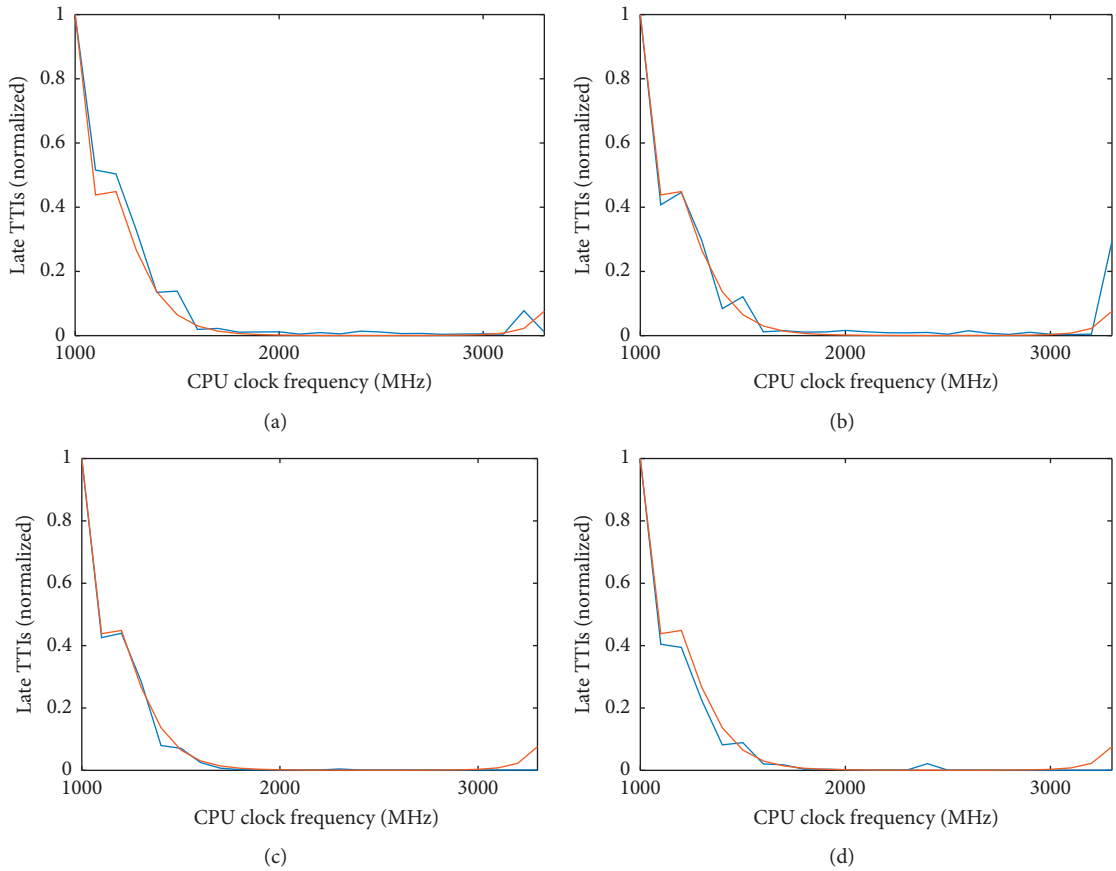

FIGURE 11: System 2: comparison of actual and predicted lates for f_0 with seven parameters f_m. (a) 5 MHz bandwidth. (b) 10 MHz bandwidth. (c) 15 MHz bandwidth. (d) 20 MHz bandwidth.

environment but also non-real-time data transfers. Samples must be moved to and from the radio frontends to the virtualized BBU. Network traffic interrupts were observed to have a significant impact on platform performance. Network traffic originated interrupts can disrupt the dataflow processing of the physical layer. This will not be visible in the CPU usage of the baseband task but can still cause processing to be late. CPU clock frequency

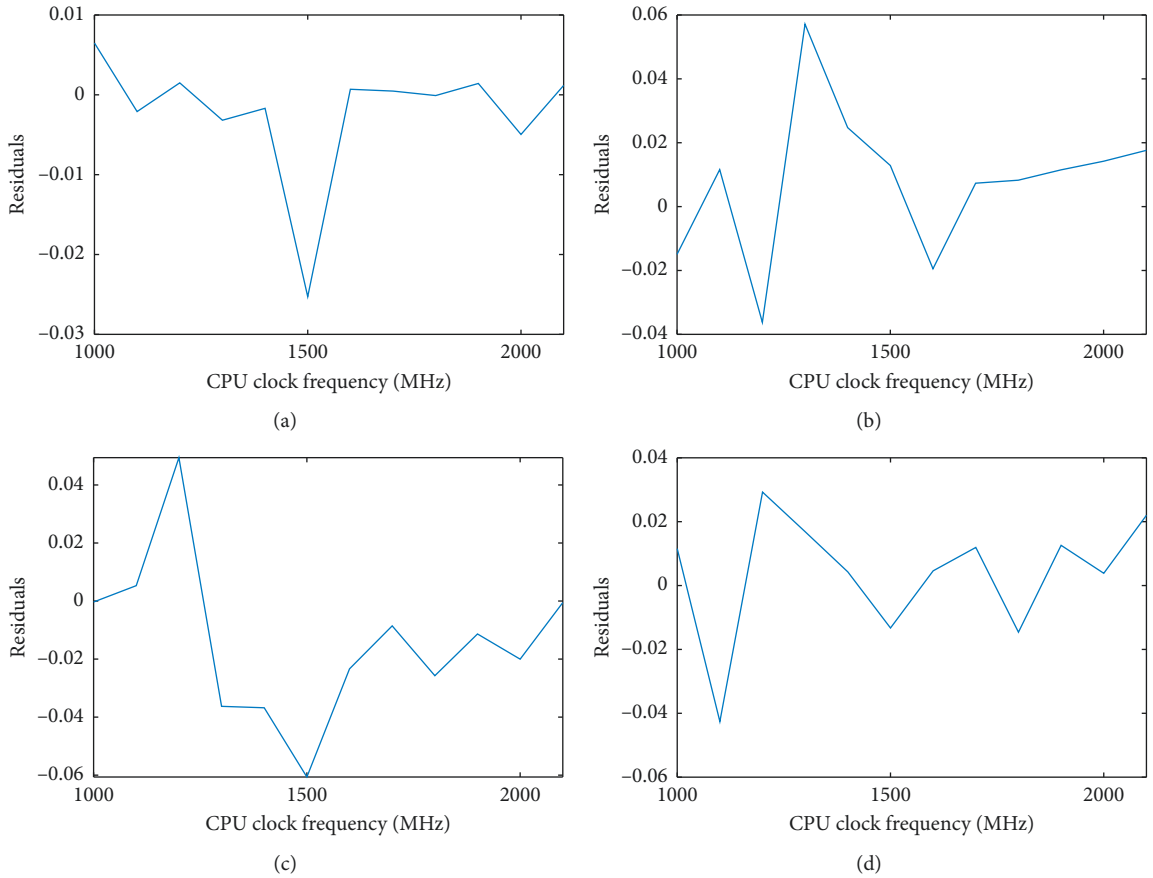

Figure 12: System 1: residuals for the predicted lates for f_m. (a) 5 MHz bandwidth. (b) 10 MHz bandwidth. (c) 15 MHz bandwidth. (d) 20 MHz bandwidth.

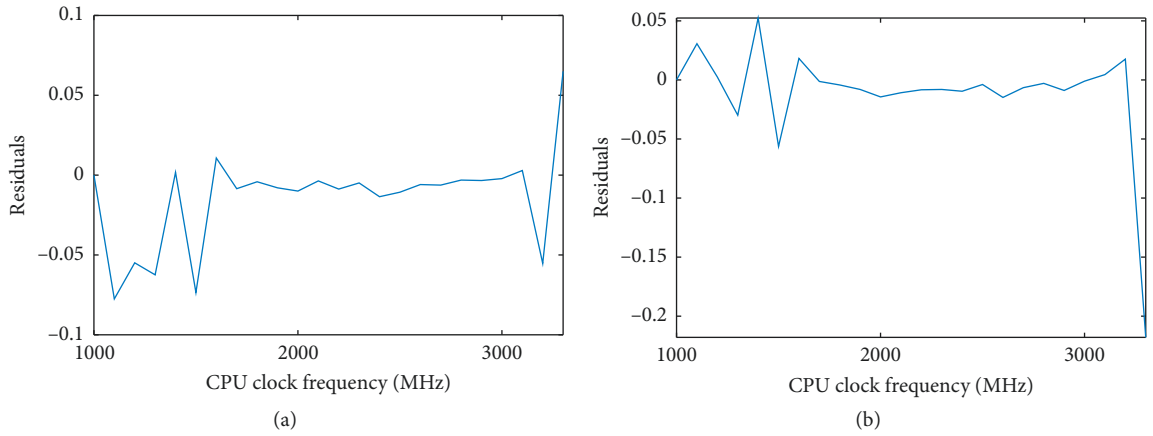

Figure 13: Continued.

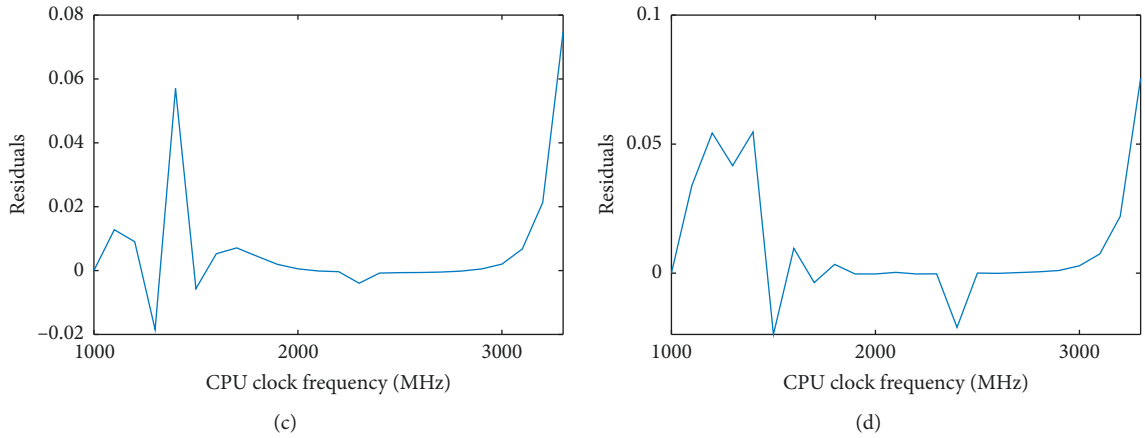

Figure 13: System 2: residuals for the predicted lates for f_0 with seven parameters. (a) 5 MHz bandwidth. (b) 10 MHz bandwidth. (c) 15 MHz bandwidth. (d) 20 MHz bandwidth.

adjustment algorithms may therefore make incorrect predictions. A smaller standard deviation provides a more predictable environment for the SDR platform to adapt to, thus making it possible to scale CPU frequency down more aggressively.

Measurements were carried out to quantify the proportion of time spent handling network related functions. In an effort to isolate variance to that caused by network processing itself, no LTE processing was executed concurrently. Tables 9 and 10 show round-trip time (RTT) obtained for 50 megasamples per second with 368 samples per TTI at two different MTU sizes.

One method to relieve the CPU is the use of offload-capable NICs. These take on the task of processing incoming network packets instead of the OS. As such, they only deliver ready-to-use payload data straight to the application as there is no need to invoke any OS kernel functionality. Offloading network processing presents gains orthogonal and independent of baseband processing capacity.

In both the small and large MTU cases, offloaded network processing provided the best performance. The much lower standard deviation improves reliability of the BS software when the CPU clock frequency is adjusted close to the minimum required as described in Section 8.1. These latencies measured represent a lower bound on TTI preparation time that is independent of computing capacity. It should also be noted that the offloading results in less CPU usage and context switches.

While LTE requirements can be met, the impact of network processing on SDR performance becomes more severe as TTI durations decrease. In 5G, TTI duration is expected to shrink to provide lower latency communication. The ratio of TTI duration to interrupt handling latency will worsen the risk becoming the limiting factor for 5G SDR implementations. Additionally, larger bandwidth will require more samples to be transferred between RRHs and BBU. The resultant increase in network traffic increases the number of packets to process and thus network processing load on the OS.

Table 9: System 3: latency results per NIC configuration. Sampling rate 50 Msps, 368 samples per TTI, and MTU of 604 bytes.

Test configuration	Average latency (μs)	Standard deviation
Mellanox 40G kernel	171.8	90.7563
Mellanox 40G offload	59.8	14.7207
Intel 40G kernel	127	42.4676

Table 10: System 3: latency results per NIC configuration. Sampling rate 50 Msps, 368 samples per TTI, and MTU of 8028 bytes.

Test configuration	Average latency (μs)	Standard deviation
Mellanox 40G kernel	155.8	34.3977
Mellanox 40G offload	119.4	8.5615
Intel 40G kernel	181.8	63.4287

9. Conclusion

Cellular BS execution on general purpose hardware and OSs is feasible. The SDR aim of increasing the flexibility of communication protocol implementations can be met if system parameters are appropriately taken into account to achieve optimal performance. It was demonstrated that acceptable performance can be obtained without resorting to a hard-real-time OS or dedicated ASIC-based designs. This opens up the possibility of consolidating BS processing into C-RAN data centers. This could further increase the flexibility of SDR system deployment by enabling on-demand allocation of resources to those cells experiencing load while activating power saving measures on unloaded ones. Concentration of computations produces savings in the hardware investments required to build the network. In addition, energy savings can be realised through CPU frequency scaling. Measurements showed that the relationship between operating frequency and the probability of late subframe processing is nonlinear. By operating at, or slight above, the threshold point, performance can be kept at virtually the same level as fully on but with a significantly lower energy consumption.

It was shown that optimising for computations-per-second is not sufficient—latencies and jitter caused by competing processes must be taken into account. On the other hand, energy-efficient use of computing resources demands that CPU clock frequencies be kept low and idle periods extended as much possible. In order to combine these two objectives—few missed TTI deadlines and high power savings—parameter tuning should take into account not only the processing needs of the application itself but also delays external to the platform. In particular, latency from passing data from the NIC through the network stack to the user process was studied. It was found that network traffic generated interrupts impact processing time and must therefore also be taken into consideration. Especially for shorter TTI durations, network processing delay can constitute a substantial portion of the total computation time budget. Accounting for this is of particular relevance in C-RAN environments, where RRHs are physically separate from the servers executing the baseband processing. Mitigating the impact of moving data to-and-from BBU constitutes an important avenue of research to improve the performance of virtualized cellular software platforms. NICs offering full network stack offload constitute one possible solution.

Several metrics along with a performance model were also proposed to help in estimating suitable operating points for the CPU clock frequency. Such a model may be used to determine appropriate resource allocation levels for software-based BSs according to load. Design of a self-tuning platform is left up to future work.

Data Availability

The latency measurement data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded in part by Business Finland under the project "TAKE-5: 5th Evolution Take of Wireless Communication Networks" and the PriMO-5G project funded by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 815191. Article processing charges were paid by the Primo-5G Project.

References

[1] Z. Chen and J. Wu, "LTE physical layer implementation based on GPP multi-core parallel processing and USRP platform," in *Proceedings of the 2014 9th International Conference on Communications and Networking in China*, pp. 197–201, Maoming, China, August 2014.

[2] H. Shen, X. Wei, H. Liu, Y. Liu, and K. Zheng, "Design and implementation of an LTE system with multi-thread parallel processing on OpenAirInterface platform," in *Proceedings of the 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, Montreal, Canada, September 2016.

[3] Z. Geng, X. Wei, H. Liu, R. Xu, and K. Zheng, "Performance analysis and comparison of GPP-based SDR systems," in *Proceedings of the 2017 7th IEEE International Symposium on Microwave, Antenna, Propagation, and EMC Technologies (MAPE)*, pp. 124–129, Xi'an, China, October 2017.

[4] P. Rost, I. Berberana, A. Maeder et al., "Benefits and challenges of virtualization in 5G radio access networks," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 75–82, December 2015.

[5] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: architectures, challenges, and approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495–2508, 2018.

[6] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818–1831, 2017.

[7] G. Rittenhouse, S. Goyal, D. T. Neilson, and S. Samuel, "Sustainable telecommunications," in *Proceedings of the 2011 IEEE Technical Symposium at ITU Telecom World (ITU TW)*, pp. 19–23, Geneva, Switzerland, October 2011.

[8] IEEE, *IEEE Std 1900.1-2008: IEEE Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management*, IEEE, Piscataway, NY, USA, 2008.

[9] P. D. Francesco, S. McGettrick, U. K. Anyanwu, J. C. O'Sullivan, A. B. MacKenzie, and L. A. DaSilva, "A split architecture for random access MAC for SDR platforms," in *Proceedings of the 8th International Conference on Cognitive Radio Oriented Wireless Networks*, pp. 250–255, Washington, DC, USA, July 2013.

[10] Y. S. Kuo, T. Schmid, and P. Dutta, "Demo abstract: a compact, inexpensive, and battery-powered software-defined radio platform," in *Proceedings of the 2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 137-138, Beijing, China, April 2012.

[11] X. Wang, S. Thota, M. Tornatore et al., "Energy-efficient virtual base station formation in optical-access-enabled cloud-RAN," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1130–1139, 2016.

[12] S. S. Kumar and A. Kumar, "Energy efficient rate coverage with base station switching and load sharing in cellular networks," in *Proceedings of the 2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–6, January 2016, Bangalore, India.

[13] E. Oh, K. Son, and B. Krishnamachari, "Dynamic base station switching-on/off strategies for green cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 5, pp. 2126–2136, 2013.

[14] P. Kolios, V. Friderikos, and K. Papadaki, "Switching off low utilization base stations via store carry and forward relaying," in *Proceedings of the 2010 IEEE 21st International Symposium on Personal, Indoor and Mobile Radio Communications Workshops*, pp. 312–316, Instanbul, Turkey, September 2010.

[15] D. Mishra, P. C. Amogh, A. Ramamurthy, A. A. Franklin, and B. R. Tamma, "Load-aware dynamic RRH assignment in cloud radio access networks," in *Proceedings of the 2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, Doha, Qatar, April 2016.

[16] L. Jun, L. Tingting, C. Gang, Y. Hua, and L. Zhenming, "Mining and modelling the dynamic patterns of service providers in cellular data network based on big data analysis," *China Communications*, vol. 10, no. 12, pp. 25–36, 2013.

[17] I. C. Bertolotti, "Real-time operating systems tutorial," in *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics*, pp. 4023–4120, Bari, Italy, July 2010.

[18] R. Kaiser, "Alternatives for scheduling virtual machines in real-time embedded systems," in *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, pp. 5–10, ACM, Glasgow, Scotland, April 2008.

[19] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*, pp. 598–610, ACM, New York, NY, USA, December 2015.

[20] C.-H. Chou, D. Wong, and L. N. Bhuyan, "Dynsleep: fine-grained power management for a latency-critical data center application," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED '16)*, pp. 212–217, ACM, San Francisco, CA, USA, August 2016.

[21] X. Zhong and C.-Z. Xu, "Frequency-aware energy optimization for real-time periodic and aperiodic tasks," *ACM SIGPLAN Notices*, vol. 42, no. 7, pp. 21–30, 2007.

[22] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Communications of the ACM*, vol. 60, no. 4, pp. 48–54, 2017.

[23] X. Tao, Y. Hou, H. He, K. Wang, and Y. Xu, "GPP-based soft base station designing and optimization (invited paper)," in *Proceedings of the 2012 7th International Conference on Communications and Networking in China*, pp. 49–53, Kun Ming, China, August 2012.

[24] P. Guo, X. Qi, L. Xiao, and S. Zhou, "A novel GPP-based software-defined radio architecture," in *2012 7th nternational Conference on Communications and Networking in China*, pp. 838–842, Kun Ming, China, August 2012.

[25] Z. Wang, L. Xiao, X. Su, X. Xu, and X. Qi, "On the optimization of real time performance of software defined radio on Linux OS," *Communications and Network*, vol. 5, no. 3, pp. 292–297, 2013.

[26] 3GPP, *LTE; Evolved universal terrestrial radio access (E-UTRA); Physical Layer Procedures (3GPP TS 36.213 Version 8.8.0 Release 8)*, ETSI, Sophia Antipolis, France, 2009.

[27] A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Implementation of a fully-parallel turbo decoder on a general-purpose graphics processing unit," *IEEE Access*, vol. 4, pp. 5624–5639, 2016.

[28] J. Kerttula, N. Malm, K. Ruttik, R. Jäntti, and O. Tirkkonen, "Implementing TD-LTE as software defined radio in general purpose processor," in *Proceedings of the 2014 ACM Workshop on Software Radio Implementation Forum*, pp. 61–68, Chicago, IL, USA, August 2014.

[29] UHD, Ettus Research, 2019, https://files.ettus.com/manual/.

[30] Intel Corporation, *Intel Xeon Processor D-1541*, Intel Corporation, Santa Clara, CA, USA, 2017, http://ark.intel.com/products/91199/Intel-Xeon-Processor-D-1541-12M-Cache-2_10-GHz.

[31] Intel Corporation, *Intel Xeon Processor E3-1230v3*, Intel Corporation, Santa Clara, CA, USA, 2017, http://ark.intel.com/products/75054/Intel-Xeon-Processor-E3-1230-v3-8M-Cache-3_30-GHz.

[32] Intel Corporation, *Intel Xeon Processor E5-1650v4*, Intel Corporation, Santa Clara, CA, USA, 2017, https://ark.intel.com/products/92994/Intel-Xeon-Processor-E5-1650-v4-15M-Cache-3_60-GHz.

[33] USRP N200, Ettus Research, 2016, https://www.ettus.com/product/details/UN200-KIT.

[34] USRP N200, Ettus Research, 2017, https://www.ettus.com/product/details/X300-KIT.

[35] P. Regnier, G. Lima, and L. Barreto, "Evaluation of interrupt handling timeliness in real-time Linux operating systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 52–63, 2008.

[36] F. M. David, J. C. Carlyle, and R. H. Campbell, "Context switch overheads for Linux on ARM platforms," in *Proceedings of the 2007 Workshop on Experimental computer science (ExpCS '07)*, San Diego CA, USA, June 2007.

[37] A. Tsariounov, *(2008–2011) cpuset*, Novell Inc., Provo, UT, USA, 2016, https://github.com/lpechacek/cpuset.

[38] J. Zhang, X. Lu, and D. K. Panda, "Performance characterization of hypervisor-and container-based virtualization for HPC on SR-IOV enabled InfiniBand clusters," in *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1777–1784, Chicago, IL, USA, May 2016.

[39] R. Householder, S. Arnold, and R. Green, "Simulating the effects of cloud-based oversubscription on datacenter revenues and performance in single and multi-class service levels," in *Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing*, pp. 562–569, Anchorage, AK, USA, June 2014.

[40] C. Chou, L. N. Bhuyan, and D. Wong, "μDPM: Dynamic power management for the microsecond era," in *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 120–132, Washington, DC, USA, Febuary 2019.

[41] M. Arora, S. Manne, Y. Eckert, I. Paul, N. Jayasena, and D. Tullsen, "A comparison of core power gating strategies implemented in modern hardware," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1, pp. 559-560, 2014.

[42] S. D. Lembo, "Modeling BLER performance of punctured turbo codes," diplomityö, Pirkkala, Finland, 2011, http://urn.fi/URN:NBN:fi:aalto-201207022672G2 Pro gradu.

[43] Mathworks, *(1994–2018) Matlab*, Mathworks, Natick, MA, USA, 2018, https://se.mathworks.com/products/matlab.html.