CloudIQ: A Framework for Processing Base Stations in a Data Center

Sourjya Bhaumik[1], Shoban Preeth Chandrabose[1], Manjunath Kashyap Jataprolu[1], Gautam Kumar[2], Anand Muralidhar[1], Paul Polakos[3], Vikram Srinivasan[1], and Thomas Woo[4]

[1]Alcatel-Lucent Bell Labs, India

[1]{sourjya.bhaumik, shoban.preeth_chandrabose, manjunath_kashyap.jataprolu, anand.muralidhar, vikram.srinivasan}@alcatel-lucent.com

[2]U. C. Berkeley

[2]gautamkumar.iit@gmail.com

[3]Cisco, USA

[3]ppolakos@cisco.com

[4]Alcatel-Lucent Bell Labs, USA

[4]thomas.woo@alcatel-lucent.com

ABSTRACT

The cellular industry is evaluating architectures to distribute the signal processing in radio access networks. One of the options is to process the signals of all base stations on a shared pool of compute resources in a central location. In this centralized architecture, the existing base stations will be replaced with just the antennas and a few other active RF components, and the remainder of the digital processing including the physical layer will be carried out in a central location. This model has potential benefits that include a reduction in the cost of operating the network due to fewer site visits, easy upgrades, and lower site lease costs, and an improvement in the network performance with joint signal processing techniques that span multiple base stations. Further there is a potential to exploit variations in the processing load across base stations, to pool the base stations into fewer compute resources, thereby allowing the operator to either reduce energy consumption by turning the remaining processors off or reducing costs by provisioning fewer compute resources. We focus on this aspect in this paper.

Specifically, we make the following contributions in the paper. Based on real-world data, we characterise the potential savings if shared homogeneous compute resources are used to process the signals from multiple base stations in the centralized architecture. We show that the centralized architecture can potentially result in savings of at least 22% in compute resources by exploiting the variations in the processing load across base stations. These savings are achievable with statistical guarantees on successfully processing the base station's signals. We also design a framework that has two objectives: *(i)* partitioning the set of base stations into groups that are simultaneously processed on a shared homogeneous compute platform for a given statistical guarantee, and *(ii)* scheduling the set of base stations allocated to a platform in order to meet their real-time processing requirements. This partitioning and scheduling framework saves up to 19% of the compute resources for a probability of failure of one in 100 million. We refer to this solution as CloudIQ. Finally we implement and extensively evaluate the CloudIQ framework with a 3GPP compliant implementation of 5 MHz LTE.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems,Signal processing systems*

General Terms

Algorithms, Design

Keywords

Cellular, Virtualization, Cloud RAN

1. INTRODUCTION

The proliferation of devices like tablets and smart phones, coupled with new types of applications, has resulted in a demand for high data rates in wireless communication. There are several innovative proposals to meet this demand for the explosion in wireless capacity [1]. These proposals require new architectures, protocols, and advanced signal processing techniques in a cellular base station. However, implementing these proposals increases the cost of delivering data to end users. Our approach is motivated by a proposal called Cloud Radio Access Network (Cloud RAN) that proposes a transition from distributed to centralized infrastructure for baseband processing of cellular signals. This is under consideration by NGMN [3]. Cloud RAN can potentially reduce the cost of delivering data and also enable signal processing techniques that improve network performance. Preliminary studies by China Mobile show that Cloud RAN has the potential to significantly reduce the total cost of operations [2].

In existing cellular networks, compute resources for performing baseband processing are located at each cell site. However, in the proposed centralized architecture, the compute resources are located in a central location. The architecture is realized by transporting the cellular signals (also known as IQ or In phase-Quadrature phase signals) over dedicated high speed fiber from the antennas at the cell site to the central location. The cell sites are situated in a radius of up to 20 km [3] from the central location. There are potential benefits to this approach which include (i) a savings in the operating expenses due to a reduction in the site visits for upgrade and maintenance, and possibly lower site lease costs, (ii) use of advanced coordinated signal processing techniques by jointly processing signals from neighboring base stations to significantly improve network performance [16], and (iii) a potential reduction in energy expenditure by exploiting the load variations and using only as many compute resources as required to process the base stations. This would exploit the variations in the processing load of base stations to multiplex multiple base stations on to a single compute resource. Energy reduction is a key consideration for operators in Asia. The operational expenditure from energy costs can be as high as 50% of the total operational expenditure in these regions. However the costs of transporting the IQ signals in the centralized architecture need to be factored in while evaluating the advantages of centralized processing, but this economic analysis is outside the scope of this paper. To address this, the cellular industry is actively considering some split processing alternatives, where some of the Layer 1 processing is done at the cell site and only user related Layer 1 processing is sent to the central location. This has the potential to significantly reduce the backhaul requirements and make it proportional to user load. Our goal in this paper is to evaluate the third benefit of the centralized architecture, i.e., we study the benefits of resource pooling across base stations and develop a framework to multiplex multiple base stations on a platform with a set of homogeneous compute resources. We refer to our approach as CloudIQ. CloudIQ is further motivated by emerging trends towards a more programmable wireless infrastructure. For example, many equipment manufacturers are migrating towards making the hardware more programmable and flexible by combining multiple hardware, DSPs, and programmable processor cores into single die (System on a Chip), while still maintaining the required cost/performance ratios. Further there is evidence that implementing wireless infrastructure on a general purpose processor (GPP) based platform is feasible for WiFi access points and LTE base stations [21]. Although GPP-based systems do not meet the current requirements in terms of performance-per-Watt, we evaluate the CloudIQ architecture on a homogeneous GPP-based platform since it provides a simple model to analyze the resource management problem and has wider applicability to resource sharing among a set of homogeneous compute resources (e.g. FPGAs, DSPs, or ASICs).

1.1 Objectives and Contributions

There have been several reports of trials of the Cloud radio access network concept by various vendors, to the best of our knolwedge, these demonstrations only backhaul the IQ samples to a central location and the rest of the processing is done as in existing cellular systems, i.e., each base station is assigned a single base band processing unit and base stations are not pooled on fewer baseband units to exploit energy savings or reduce the number of processing resources. Our paper is the first to do a systematic study of the gains possible due to resource pooling gains and provide a framework to precsiely achieve a trade-off between the statistical guarantees against failure and the savings (energy/cost) possible due to the Cloud RAN architecture. We elaborate on the contributions below.

1. **Analyze the benefits of CloudIQ on general compute platforms:** In traditional cellular systems, the compute resources are provisioned to handle the peak load at a base station, while meeting the real-time deadlines. However, in a CloudIQ architecture, the variations in the processing loads of base stations can be used to multiplex multiple base stations on a single compute resource. The variations in the load happen at small time scales (order of ms), but it is not possible to change the assignment of base stations to compute resources at such time scales. Hence, we allow changes in assignment of base stations to compute resources at larger time scales (order of minutes) and provide statistical guarantee on the base stations successfully meeting their real-time deadlines. We analyze real-world WCDMA data to study the feasibility and advantages of this approach. This data contains PHY layer logs from 175 base stations in a dense urban setting. We observe that, for realistic statistical guarantees, resource savings of at least 22% are possible in the CloudIQ architecture. See Section 3 for the details.

2. **Resource management framework:** In order to realize the potential savings of compute resources, we need to process multiple base stations in a single compute resource and meet their individual real-time constraints. We refer to this as *virtualization* of the base stations. There is a wealth of literature on scheduling of real-time systems, which includes scheduling on multi-processor platforms. But most of the existing work considers systems with *implicit deadlines* where the deadline coincides with the periodicity of the tasks. As we see later, this is unlike cellular systems where the deadline to process a task can exceed the periodicity of the task. Furthermore, we adopt a *separation principle* in the design of the system that adds to its simplicity and robustness. The separation principle decouples the problem of partitioning the base stations into subsets to be scheduled on each compute resource from the problem of designing algorithms to schedule a subset of base stations on a compute resource and meet their deadline constraints.

We design a resource management framework that adheres to the separation principle and can schedule multiple base stations on a homogeneous compute platform. Each homogeneous compute platform consists of multiple identical cores, where each core is equally capable of executing a task. This framework is simple to implement and the savings in the compute resources under the framework are close to those obtained by analyzing the real-world data. See Section 4 for the details of the framework.

3. **System Design and Evaluation:** We implement the CloudIQ framework on an Intel Xeon six core processor, where each core has a clock speed of 3.47GHz. We choose Linux with PREEMPT_RT patch as the operating system. We choose an open source implementation of the LTE standard called OpenAir to implement and demonstrate the resource management framework. We evaluate our implementation and show that two 5 MHz LTE base stations at peak load can be run on a single processor, while meeting their individual real time requirements. Peak load refers to the maximum load in processing the physical layer in 5 MHz LTE. We can schedule a maximum of four base stations in a single platform and meet their real-time requirements if their individual loads are below the peak processing load. See Section 5 for the details of the system implementation.

Section 2 contains a short primer on cellular systems and describes the OpenAir project. We overview the related work in Section 7 and outline some of the remaining challenges in Section 8.

2. PRELIMINARIES

2.1 Primer on cellular systems

In a cellular system, spectrum is treated as a resource that is shared among all the users. The spectrum is partitioned into *channel resources*, which are spreading codes in WCDMA or time-frequency resources referred to as physical resource blocks (PRB) in LTE. In each scheduling interval, the scheduler allocates certain number of channel resources and specifies a modulation and coding scheme (MCS) for a user to either receive or transmit data. As per LTE the smallest granularity for scheduling decisions is 1 ms, while this is much higher in WCDMA. In each scheduling decision epoch, the scheduled users transmit (receive) data in fixed time intervals called subframes. In LTE, a subframe has a duration of 1 ms, while in WCDMA, a subframe lasts for 2 ms. Each transmitted subframe is acknowledged (ACKed) in a subsequent subframe. In FDD-LTE, the ACK for subframe k is sent in subframe $k + 4$. Since the subframes need to be decoded before an ACK is sent, the receiver in LTE has 3 ms to decode the subframe, while the receiver in WCDMA has more time to decode the subframes.

Observation 1. *In cellular systems, physical layer processing jobs arrive periodically every T units of time. The deadline d for processing a job is typically larger than T.*

2.2 OpenAir Project

We use an open source implementation of LTE called OpenAir [5] to demonstrate the CloudIQ framework that virtualizes base stations. This was developed by the OpenAir project and is a 3GPP compliant 5 MHz LTE implementation. We choose an implementation of LTE over readily available implementations of WiFi (for example, [21]) because our primary goal is to virtualize cellular systems. Also, cellular systems like LTE have a synchronous deterministic structure, while WiFi is asynchronous and data packets can be transmitted or received at arbitrary times.

2.2.1 *Profiling of OpenAir*

In this section, we want to compute the processing load offered by every function in the LTE physical layer and characterise the parameters that influence the load. A block diagram of the physical layer in a LTE base station is shown in Figure 1. We refer the interested reader to [1] for a detailed description of the various blocks in the figure. The key observation from this figure is that the Fast Fourier transform (FFT) and inverse FFT (IFFT) functions on the downlink and uplink chains respectively are performed in each subframe and are independent of the number of allocated PRBs or MCS. These functions impose a constant base processing load on the system. All the functions subsequent to the FFT or the IFFT depend on PRBs and the MCS allocated to users and the channel conditions.

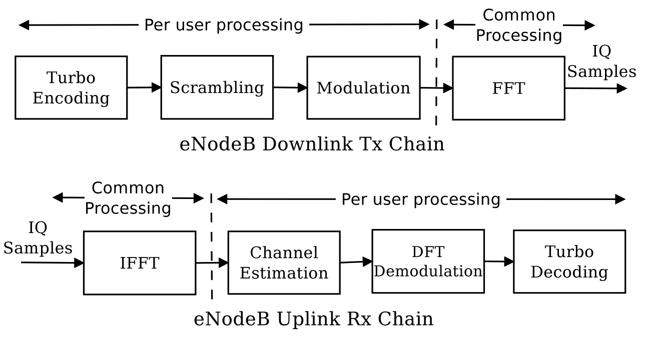


Figure 1: LTE base station PHY processing blocks

We wish to profile the physical layer of LTE to understand the relationship between the processing load and number of allocated PRBs and MCS. We study this relationship by scheduling a single user for transmission or reception. As we see later, the profiling results allow us to generalize the observations to multiple users. We profile the physical layer of the OpenAir implementation of LTE by varying the following parameters: *(i)* the number of PRBs used, *(ii)* the MCS, and *(iii)* the SINR for a given MCS value[1]. We also calculate the time taken for execution by varying these parameters for the downlink and the uplink scenarios.

Based on the profiling results in Figure 2, we make the following observations.

1. *The uplink dominates the downlink processing load and is about* 2.5 *times the downlink processing load for a given MCS, see Figure 2(a).* As shown in Figure 2(c), the uplink processing of a subframe is largely dominated by the turbo decoder, while the remaining functions offer a constant load. The turbo decoder is an iterative algorithm where every iteration improves the decoder's estimate of the transmitted codeword (see [8] for details). We can see from Figure 2(b) that there is no function in the downlink that matches the turbo decoder in processing complexity.

2. *The processing load is well approximated as a linear function of the MCS and channel resources (PRBs), see Figures 2(b), 2(c), and 2(d). The load is divided into two parts; a constant part (base load) that is independent of MCS, PRB,*

[1]Processing time of the turbo decoder depends on the number of iterations, which is in turn a function of the channel conditions. Hence, we choose the operating SINR for a MCS so that turbo decoder converges within a bounded number (7) of iterations.

and SINR[2]*, and a dynamic part that varies linearly with the MCS and PRBs.*

3. *Since the dynamic load is linear (in fact, convex) in the MCS and the PRBs, if multiple users are simultaneously scheduled for transmission or reception, then the total processing load is given by the sum of the individual loads and cannot exceed the processing load of a single user who is allocated all the PRBs at the highest MCS.*

4. *We can generalize the observations of the LTE profiling results to other cellular standards which use different channel resources but share other components of the physical layer. This applies to WCDMA where the channel resources are spreading codes, but the MCS and the turbo encoding/decoding blocks are similar to those in LTE.*

3. ANALYSIS OF REAL-WORLD DATA

In current cellular systems, the coverage area of an operator consists of many *cell sites* and each cell site is further subdivided into three equally sized *sectors*. Each sector is assigned a set of *carriers* for transmission, where a carrier corresponds to a band of frequencies in the spectrum assigned to the carrier. There is a limit on the number of users that can be supported by a carrier and sectors that experience high load are assigned multiple carriers. Therefore, a cell site can consist of many *sector-carriers*. In the rest of the section, we refer to a sector-carrier with the more commonly used term; *base station*.

In the traditional setup, each base station is provided with computing resources for performing the signal processing operations in cellular communications. This critical infrastructure is called the baseband processing unit (BBU). A cell site with multiple base stations has a separate BBU dedicated to each base station. The BBU is provisioned to handle the peak load in a base station. Our goal in this paper is to break away from this model and study the advantages of *pooling computing resources across base stations*. We allow sharing of a common pool of homogeneous compute resources across base stations and facilitate multiple base stations to perform their signal processing operations in a single computing resource. Since CloudIQ differs from the traditional approach, we do not refer to the compute resource as a BBU to avoid confusion. In this section, we use logs of real-world traffic to characterise the processing load distribution at a base station. Then, we study the potential savings in the number of compute resources in the CloudIQ architecture as a function of the processing load distributions.

3.1 Real-world cellular traffic

We obtained detailed logs of traffic in a WCDMA network from a cellular operator across 21 cell sites in a dense urban setting. We choose WCDMA over LTE, because WCDMA is widely deployed while LTE adoption is still in its infancy. Hence WCDMA logs are more likely to reflect true traffic patterns. We use these logs to estimate processing loads at base stations and the effect of resource pooling across different base stations. We analyze only processing time distributions for downlink traffic, i.e., transmissions from the base station to the mobile user. The logs had insufficient information on the traffic in the uplink direction.

[2]Independence of the base load from SNR is not shown in the plots in Figure 2 due to lack of space.

Each of the 21 cell sites has 4 to 12 sector-carriers (base stations). In total, we have 175 base stations spread over the cell sites. A carrier in WCDMA corresponds to a bandwidth of 5 MHz. The data is aggregated across 15 minute intervals. This is the minimum granularity at which we obtained logs. In WCDMA, there are 450000 sub-frames in a 15 minute interval and there are a total of 15 spreading codes which are shared between voice and data traffic. A spreading code cannot be simultaneously used for voice and data. The logs are solely based on mobile data usage and not voice.

The logs give us the following three pieces of information for each 15 minute interval:

- Total downlink data sent from the base station to the mobile users, in bits.
- Number of sub-frames in which each QAM modulation (QPSK, 16-QAM, or 64-QAM) is used in downlink.
- The number of spreading codes *available* for data traffic in each scheduling interval. Note that all the available codes *need not* be used for data transmission.

We face a challenge in using this aggregate logs at each base station to estimate the processing load distribution in a subframe. This distribution can be potentially different across time intervals (e.g., peak hours in the morning vs. late in the night). Our goal is to obtain a conservative estimate of the processing load distribution and avoid overstating benefits of the CloudIQ framework. In order to derive these estimates, we use the following steps:

1. We use the logs available on QAMs to estimate the probability that a particular QAM is used in a subframe.

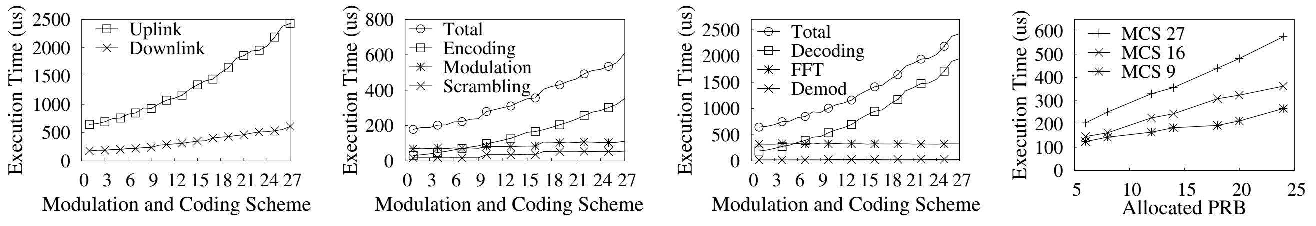
2. Similarly, we use the logs on spreading codes *available* to estimate the probability that a certain number of codes are available in a subframe.

3. Next, we use the observation in Section 2 that the processing load is linear in the MCS and channel resources to estimate the distribution on the processing load. From the previous step, we know that this estimate is in fact an overestimate of the processing load since all the available codes need not be used for transmission.

4. In order to fix the issue of overestimating the processing load, we use the logs on the total traffic in the downlink direction in a 15 minute interval. Now, with the computed distributions of QAMs and available codes, we estimate the total traffic in the same 15 minute interval. We compare the estimated traffic with the actual traffic and use the difference between them to scale down the distribution on the codes available per subframe in the 15 minute interval. This further leads to a scaling down of the processing load distribution.

The details of this are outlined in Section 3.2.

In the steps outlined above, we make a crucial assumption that the distributions of QAMs and channel resources used in 5 MHz WCDMA is same as that in 5 MHz LTE. Although there are a few salient differences between WCDMA and LTE in terms of the sub-frame durations and the channel resources, our assumption is justified for the following reasons. The distribution of QAM modulations is a function of the relative location of the users with respect to the base station. Users close to the base station experience a good



(a) Comparison of uplink and downlink processing times

(b) Downlink processing times

(c) Uplink processing times

(d) Downlink processing times as a function of PRBs

Figure 2: Results of profiling OpenAir code.

channel and observe high signal-to-noise-ratio (SNR), and are assigned higher order QAM modulations. Users experience signal degradation as they move away from the base station and will experience a reduction in the SNR. Consequently, they will be assigned lower order QAM modulations by the scheduler. Furthermore, the distribution of the user locations in LTE will be similar to the distribution for WCDMA as long as the cell sites and coverage areas remain the same. Hence, the same probability distribution of QAM modulations applies to both WCDMA and LTE. Similarly, the distribution of the codes across a 15 minute interval is a function of the data requirements of the users and their geographical locations. The data requirements of users may increase in LTE, but the distribution of the PRBs assigned to users in LTE will resemble the distribution of codes in WCDMA (although the absolute numbers might be higher in the next generation technology). This is because the objective of the scheduler in the base station in both the technologies is to ensure proportional fairness in the allocation of channel resources to users.

3.2 Distribution on the QAMs and codes

Since we are given the number of subframes in which each QAM modulation is used in a 15 minute interval, and there are 450000 subframes in a 15 minute interval, it is easy to compute the probability that a particular QAM is used in a subframe. Note that there are empty sub-frames where no QAM modulation scheme is used (and hence, no data was transmitted). We denote the distribution of QAMs as $(p_{\text{no-QAM}}, p_{\text{QPSK}}, p_{\text{16-QAM}}, p_{\text{64-QAM}})$. The distribution of QAMs is skewed in favor of QPSK (roughly 90%), with 16-QAM (roughly 9.9%) used less often, and 64-QAM used rarely (0.1%). Therefore the distribution of QAMs is already indicative of the fact that the typical processing load is far less than the peak processing load.

Similarly we can determine the distribution of the spreading codes available for data traffic and denote it by $(p_{\text{codes0}}, p_{\text{codes1}}, \ldots, p_{\text{codes15}})$. Recall that the log contains the codes available in each scheduling interval and can be a significant overestimate of the actual distribution of spreading codes used for transmission.

3.3 Computing distribution of processing load

In Section 2, we observed that the processing load can be split into two components: base processing load and dynamic processing load. The dynamic processing load is a function of the actual number of channel resources used in a sub-frame and MCS assigned to each channel resource. We assume that for each modulation, the highest coding rate is used in order to maximize the processing load for that modulation scheme. This also allows us to make a conservative estimate on processing loads. The corresponding MCS values for QPSK, 16-QAM and 64 QAM are 9, 16, and 27.

The number of bits transmitted per channel use under these three modulations is 2, 4, and 6 bits respectively. We translate these to workloads by assuming that 16-QAM (MCS = 16) takes twice as much dynamic processing load as QPSK (MCS = 9), and 64-QAM (MCS = 27) takes thrice as much effort as QPSK. This observation is also validated in Figure 2(b), where we can compare the processing loads for the dynamic part corresponding to turbo encoding.

We assume that QPSK modulation transmitted over one channel resource consumes 1 unit of dynamic processing load. Now, we can easily derive the processing load in any scenario. For example, 16-QAM with 4 codes will consume $2 \times 4 = 8$ units of dynamic load, while 64-QAM with 7 codes will consume $3 \times 7 = 21$ units of dynamic load. In order to compute the base processing load, we refer back to Figure 2(b) and observe that the intercept of the total load on the y axis corresponds to dynamic load at MCS = 16. By the earlier arguments, the base processing load will then equal the load of 16-QAM with 15 codes, i.e., $2 \times 15 = 30$ units. The total load for any allocation of QAMs and codes is determined by adding the base and the dynamic components and can be described by a simple formula:

$$\text{Total load} \;=\; \underbrace{\frac{(\log_2 \text{QAM})}{2} \times \text{Number of codes}}_{\text{dynamic}} + \underbrace{30}_{\text{base}}, \qquad (1)$$

where QAM can take the value 4, 16, or 64 depending on whether QPSK, 16-QAM or 64-QAM was chosen. With the above formula, we can compute a distribution on the *estimated* processing load as a function of the distribution of QAMs and available codes by assuming that the choice of QAM modulations is independent of the number of codes. Since there are 3 choices for QAM modulations and 15 choices of codes, we have 45 combinations of MCS and channel resources used, with an extra option when no data is transmitted. We denote the tuple of 46 values as $(p_{\text{est}}(0), \ldots, p_{\text{est}}(45))$.

3.4 Scaling down the processing load

We first estimate the total data traffic in a 15 minute interval with the distributions on the QAMs and codes available and show that this exceeds the actual downlink traffic. Pick a 15-minute interval for a base station. With the distribution on the QAMs, we can compute the average number of bits/sub-frame/channel use, denoted by $\bar{Q}$, as

$$\bar{Q} \;:=\; 2 \times p_{\text{QPSK}} + 4 \times p_{\text{16-QAM}} + 6 \times p_{\text{64-QAM}}. \qquad (2)$$

Here, we use the fact that QPSK, 16-QAM, and 64-QAM can send 2, 4, and 6 bits of data per channel use respectively. Next we need to determine the number of channel uses from the distribution on the codes. First calculate the average number of codes in a sub-frame as

$$\bar{C} \;:=\; 0 \times p_{\text{codes0}} + 1 \times p_{\text{codes1}} + \ldots + 15 \times p_{\text{codes15}}. \quad (3)$$

We assume that each of the 15 spreading codes occupies an equal portion of the available bandwidth, implying that each code corresponds to 5/15 MHz. A sub-frame lasts for a duration of 2 ms and the number of channel uses (s-Hz) corresponding to one spreading code in a sub-frame is

$$U \;:=\; (2 \times 10^{-3}) \times (5/15)\, 10^{6}\ \text{(s-Hz)}. \quad (4)$$

Hence, from (3) and (4), the average number of channel uses in a sub-frame in the given 15 minute interval is

$$\bar{C} \times U. \quad (5)$$

From the above equation and (2), the average number of bits per sub-frame in the given interval is

$$\bar{Q} \times \bar{C} \times U \ \text{(bits / sub-frame)}. \quad (6)$$

In a 15 minute interval, there are 450000 sub-frames, and the total number of bits transmitted in the interval is given by

$$D \;:=\; \bar{Q} \times \bar{C} \times U \times 450000 \ \text{bits}. \quad (7)$$

Hence D is the estimated downlink traffic in the chosen interval. The actual downlink traffic in the same interval is available as part of the aggregate logs and we compare it with D in figure 3. As we can see, the estimated downlink

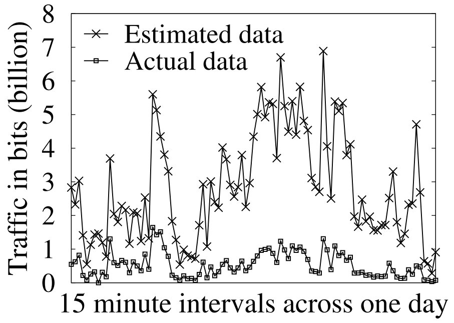


Figure 3: Comparing estimated with actual data.

traffic exceeds the actual downlink traffic. We compute the ratio $\kappa := \frac{D}{\text{Downlink traffic}}$ to capture the extent to which the estimate exceeds the actual data. The ratio κ is unique to a particular interval for a base station. We repeat this procedure across all the base stations, across all the 15 minute intervals, to compute the corresponding ratios. Clearly, we can use this information to get a better estimate of the processing load distribution. However the choice of how scaling should be done is not so straightforward.

For each 15 minute interval, κ is the scaling factor for the *average* processing load or the *first moment* of the processing load. However to compute the true processing load distribution, we will require to know the scaling factor for all the *moments* of the distribution. The scaling factor for the remaining moments is however impossible to compute with the available data. Hence there are an infinite number of processing load distributions whose average coincides with the scaled average computed earlier. Next we make a conservative estimate on the distribution. Note that the expected *estimated* processing load is given by $E_{\text{est}} = \sum_{j=0}^{45} j p_{\text{est}}(j)$. We can scale $\{p_{\text{est}}(j)\}$ in two natural ways to ensure that the new average processing load is $\frac{E_{\text{est}}}{\kappa}$. We could either scale each value of j by κ or we could scale each value of $p_{\text{est}}(j)$ by κ. Clearly scaling each processing load value j by κ will imply that the processing load never exceed $\frac{75}{\kappa}$, where 75 is the maximum processing load. This results in an optimistic estimate of the resource pooling gains. Instead, to obtain a conservative estimate, we scale the probabilities as $p_{\text{new}}(j) = \frac{p_{\text{est}}(j)}{\kappa}, j = 1, \ldots, 45$ and set $p_{\text{new}}(0) = 1 - \sum_{j=1}^{45} p_{\text{new}}(j)$ to ensure that we get a valid probability distribution.

Table 1: Reduction in aggregate computing load.

Prob. of failure (P_F)	utilization (%)	savings (%)
10^{-2}	59.99	40.01
10^{-4}	73.76	26.24
10^{-6}	76.74	23.26
10^{-8}	77.56	22.44

3.5 Gains from resource pooling

Next, we use the distribution on the processing load to calculate the aggregate computing load for base stations and check if there is a reduction in the aggregate load due to variability of the traffic across base stations. We focus on the busy hours in a day, which are from 0800 hours to 2200 hours. As mentioned before, in the traditional setup, the computing resource assigned to a base station is provisioned to handle a peak load when 64-QAM is the chosen modulation scheme and 15 codes are used for downlink transmission. This translates to a total load of 75 units. Hence, the resource can handle a peak load of 75 units in every sub-frame. As a baseline for comparison, we consider the peak load for 175 base stations, which is a total of $175 \times 75 = 13125$ units. Next, we fix a target probability of failure P_F and consider the distribution of processing loads for the base stations computed in the previous section. For each base station b, for each 15 minute interval t, we pick the maximum processing load $L_b(t)$ such that the processing load is guaranteed to be less than $L_b(t)$ for the chosen interval with probability at least $1 - P_F$. We calculate the aggregate processing load for the interval t as the sum of $L_b(t)$ for all base stations. We compute the ratio of this aggregate load with the peak load of 13125 units to compute the percentage utilization of resources when we permit statistical guarantees. We compute this ratio for different values of P_F and present the results in Table 1. We also compute percentage savings from resource pooling by subtracting the percentage consumption from 100.

We also plot the percentage utilization as a function of 15 minute intervals for the busy hours in a particular day in figure 4. As we can see from Table 1, the percentage savings

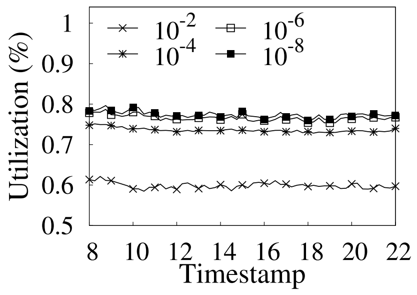


Figure 4: Utilization in busy hours.

monotonically increase with the probability of failure. Even with a very low probability of failure of 10^{-8}, we are obtain potential savings of 22.44 %. Our observations motivate us to investigate the design of virtualization mechanisms that allow multiple base stations to be pooled onto a single compute platform. Since we have made conservative estimates in calculating the processing load, we could potentially observe larger gains in practical systems.

REMARK 1. *We are modeling the cellular traffic with a probability distribution that we computed earlier, but we cannot validate this distribution based on the available real-world data. We have observed periodic patterns while computing the averages of various parameters in the available data, but extending this observation to higher order moments and to the distribution needs more fine grained data.*

4. CLOUDIQ FRAMEWORK

In this section we design the CloudIQ framework that allows pooling of compute resources across multiple base stations. Each compute resource consists of a set of identical cores. We do not assume the availability of specialized hardware as part of the platform like accelerators for computing FFT or for performing turbo decoding. In the previous section, we obtained bounds on the potential savings in the compute resources that can be obtained if we allow for statistical guarantees for the processing at base stations. In this section, our goal is two-fold: (*i*) partitioning the set of base stations into groups that are simultaneously processed on a single platform for a given statistical guarantee P_F, and (*ii*) scheduling the set of base stations allocated to a platform in order to meet their real-time constraints.

There is a wealth of literature on scheduling of real-time systems, which includes scheduling on multi-core platforms, see Section 7 for related work. But most of the work considers systems with *implicit deadline* where the deadline coincides with the periodicity of the process. This is unlike cellular systems where the deadline to process a subframe (3 ms for LTE) exceeds the periodicity (1 ms for LTE) of the subframe. Furthermore, we adopt a *separation principle* in the design of the system. We initially provision the system to handle the extreme scenario where every base station has the maximum processing load ($L_{\max}$) [3]. As we see later, this design leads to a simple solution of the above-mentioned partitioning problem with statistical guarantees. We use a standard bin-packing algorithm to pack base stations into bins of size $L_{\max}$. As a consequence of this design, the real-time processing requirements of the base stations are automatically guaranteed in the framework. This separation of the partitioning and scheduling problems is a robust and simple framework for critical real-time systems. To the best of our knowledge, such a separation is not possible under any of the existing solutions for real-time systems.

As a result of adopting the separation principle, we make the following design choices:

1. **One subframe one compute resource:** We assume that a subframe of a base station is completely processed on a single compute resource and meets its processing deadline requirement even at peak load. Different subframes of the same base station can be potentially scheduled to different compute resources. A compute resource is a logical entity and can consist of multiple physical compute resources. For example, since functions such as turbo decoding are extremely compute intensive, they may be parallelized to run across multiple physical cores.

2. **Offline schedule:** In order to adhere to the separation principle, under statistical guarantees, we allocate a certain set of base stations to a platform based on their individual processing loads. Once allocated to a platform, an offline algorithm determines the schedule of processing the subframes of the base stations and does not further optimize based on variations in the processing loads.

[3]Note that the time budget for processing is a function of the transmission delay in the network. In a fiber network, the typical transmission delay is 5μs per Km. Since the total time budget is 3 ms, the actual processing time is 3 - transmission delay ms

REMARK 2. *In this paper we do not explicitly consider signal processing techniques such as coordinated multi point scheduling (CoMP) which requires joint processing across multiple base stations. Typically however CoMP is performed across adjacent base stations and the techniques we describe here can be easily generalized to these scenarios also by constructing suitable "virtual base stations" comprising multiple physical base stations.*

4.1 Resource Management Algorithm

We develop a resource management framework that solves the partitioning and the scheduling problems mentioned above. First, we develop an algorithm to schedule base stations at peak load on a platform. This provides us with a natural framework to apply the separation principle to solve the partitioning problem. Next, under this framework, we develop an algorithm to schedule base stations with statistical guarantees. This is a standard bin packing algorithm that ensures that the base stations meet their processing deadlines.

4.1.1 A cyclic schedule

We develop a strategy that cyclically schedules base stations on a multi-core compute platform. Consider a compute platform with N cores, where the cores are ordered as $1, \ldots, N$. Consider a set of M base stations ordered as $\{B_1, B_2, \ldots, B_M\}$. The subframes at a base station arrive with a known periodicity and need to be processed within a deadline. Assume that the base stations are operating at peak load and the subframes take the maximum load $L_{\max}$ to complete processing. For the sake of exposition, let us assume that the periodicity is 1 unit while $L_{\max}$ can exceed 1. Let us view the subframes arriving for processing at a base stations as jobs and denote the j-th base station's job that arrives at time instant t as $B_j(t)$. The cyclic schedule is simple: *at time t, assign the job from the j-th base station, i.e., $B_j(t)$, to core* $(tM + j) \mod (N) + 1$.

There are two questions that immediately arise: how do we guarantee that the deadline requirements of the base stations are met and what is the maximum number of base stations that can be scheduled on a single processor. We will answer the questions next, but first we illustrate some of the properties of the cyclic schedule with an example. In the example in Figure 5, we consider a system with $N = 4$ cores which has to process $M = 3$ base stations. The subframes for each base station arrive at a periodicity of 1 unit and have a peak processing load of $L_{\max} = 4/3$ units. The resulting cyclic schedule for these parameters is shown in Figure 5. In

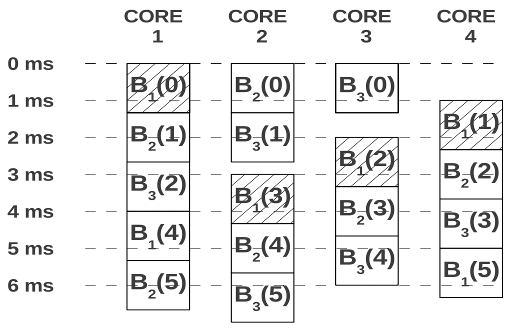


Figure 5: Example with 3 base stations and 4 cores

the figure, the horizontal axis indexes the four cores and the vertical axis indexes the time. We denote subframes by rectangles where the height of a rectangle denotes the processing load of the corresponding subframe (4/3 units).

In this example, observe the following:

- The order of the jobs processed by core 0 repeats every 4 units, it is always subframes of base station B_1 followed by B_2 followed by B_3.
- After an initial transient phase, the same schedule repeats at each core, i.e., each core sees jobs from base stations in the order B_1 followed by B_2 followed by B_3. Moreover, the jobs of a base station experience the same queuing delay for every subframe irrespective of the core they are scheduled on. In this example, the delays are 0, 1/3, and 2/3 ms for base stations B_1, B_2, and B_3 respectively.

The first property holds because $N = 3$ and $M = 4$ are coprime in the above example. If the number of cores M and the number of processors N are not coprime, it is easy to see that the sets of cores and base stations get partitioned so that each subset of cores processes a distinct subset of base stations. Let us assume that M and N are coprime for the sake of exposition. In this case, the multi-core scheduling problem essentially becomes a single core scheduling problem since the same schedule repeats across all the cores.

So far we have not dealt with the issue of synchronization between base stations at subframe boundaries. But we saw with the cyclic schedule that every base station experiences the same queuing delay for all the subframes, and this property can be used to our advantage when all the base stations are not synchronized among themselves. The subframes of different base stations need not arrive at the same time and might be delayed in relation to each other. We can handle this by closely matching the delay in receiving the subframe of a base station with the queuing delay it sees in the cyclic schedule. Revisiting the example in Figure 5, let us suppose that we have three base stations A_1, A_2, and A_3 which are not synchronized among themselves. Let us take the subframe boundaries of A_1 be the reference and assign A_1 to B_1 in the example. Then, either of A_2 or A_3 can be matched to B_2 or B_3 as long as their subframes see a relative delay of less than 1/3 and 2/3 units respectively. In the general case, the challenge is to order the base stations so that the offset in the arrival times of subframes is matched with the queuing delays and the deadline requirement of every base station is met, in other words we need to find the right permutation of base stations[4]. Also, as we later see, we can always ensure that M and N are coprime without much loss in optimality.

4.1.2 Maximizing the number of base stations

Our goal is to find the maximum number of base stations that can be scheduled on N cores, subject to the constraints that *every* subframe has the maximum load $L_{\max}$ and must be processed within a deadline $d > L_{\max}$. If a platform with N cores can schedule M base stations with load $L_{\max}$, then $ML_{\max} \leq N$ since in each unit of time the total load due to all the base stations is $ML_{\max}$. We obtain the upper bound $M \leq \lfloor \frac{N}{L_{\max}} \rfloor$ on the number of base stations. We can obtain a lower bound on M with a greedy provisioning of cores for each base station. We can assign $\lceil L_{\max} \rceil$ cores to each base station and one-by-one send subframes from the base station to each of the $\lceil L_{\max} \rceil$ cores. Clearly the deadline constraint of every base station is met with this assignment. Hence, we obtain the lower bound $M \geq \lceil \frac{N}{\lceil L_{\max} \rceil} \rceil$.

As mentioned before, if M and N are coprime, it is sufficient to analyze the schedule on a single core. We solve the scheduling problem by focusing on core 0. Let t_k denote the arrival time of the subframe for the k-th base station at core 0 and let s_k denote the delay it experiences before getting processed. As seen in the above recursion, the delay experienced by the k-th base station's subframe is 0 if the previous subframe corresponding to the $k-1$-th base station was processed by t_k. Otherwise, the delay induced due to the previous job is given by the difference between the time when the $k-1$-th job finishes and t_k. With a recursive computation, the $k-1$-th job finishes by $t_{k-1} + s_{k-1} + L_{max}$. Hence the delay experienced by the k-th base station is:

$$
\begin{aligned}
s_k &= \max(0, s_{k-1} + L_{\max} - (t_k - t_{k-1})), && (8)\\
&\quad k = 1, \ldots, M-1 \\
s_0 &= 0. && (9)
\end{aligned}
$$

The constraints that have to be satisfied at each core under our framework can be written as follows:

$$
\begin{aligned}
\text{Maximize} \quad & M \\
\text{subject to} \quad & s_k + L_{\max} \leq d, \\
& s_{M-1} + L_{\max} \leq t_M - t_{M-1}, \\
& \lceil \frac{N}{\lceil L_{\max} \rceil} \rceil \leq M \leq \lfloor \frac{N}{L_{\max}} \rfloor, \\
\text{where} \quad & t_k = \lfloor \frac{kN}{M} \rfloor, M \in \mathcal{Z}
\end{aligned}
$$

where the fist constraint ensures that the deadline constraint is met for every base station. The second constraint ensures that the last base station B_M completes processing before the first base station B_1 in the the next cycle arrives. The last constraints are the previously obtained bounds on M.

It is easy to solve this optimization. We choose M to equal the upper bound on the number of base stations. We check if the constraints are satisfied for this choice, else we reduce M by 1 and repeat the check. We stop at the first case when all constraints are met and declare it to be the maximum number of base stations that can be accommodated. If the choice of M is not coprime with N, then we further reduce M till the result is coprime with N. The probability that both M and $M-1$ are not coprime with N is less than 2% for $M, N \leq 1000$, hence the loss in optimality is very low. Hence we can always ensure that M and N are coprime.

[4]Since the duration of each subframe is 1 ms, the relative delay between the arrival of subframes of two base stations is bounded by 0.5 ms.

4.1.3 *Resource Pooling*

Recall that in order to enable a centralized RAN, we have a two-fold goal of partitioning and scheduling base stations with statistical guarantees. In Section 4.1.1, we developed a framework to schedule base stations on a multi-core platform and computed the maximum number M of base stations that a platform with N cores can support. In this section, we solve the problem of partitioning the base stations into subsets, each of which can be scheduled on an N core platform. Each subset can contain more than M base stations. We will see that the cyclic schedule allows us to apply the separation principle and deadlines for all the base stations can be met with no additional effort.

We assume that the CDF of the processing load for base station B_i at time t is known and denoted by $F_{i,t}(x)$. Let P_F be the chosen statistical guarantee or the probability of failure. We compute the processing load $L_{i,t}(P_F)$ such that the probability that the base station's actual load exceeds $L_{i,t}(P_F)$ is less than P_F:

$$L_{i,t}(P_F) \; := \sup_{x:x\leq P_F} \{F_{i,t}^{-1}(x)\}, \forall\, i,t. \tag{10}$$

Next, we need to partition this set of $\{L_{i,t}\}_i$ for a given t, into smaller sets that can be scheduled on a GPP platform. Since each of the subsets is scheduled on a separate platform, we want to minimize the number of subsets that we create. We adopt the following approach for it: recall that in Section 4.1.2, we determined the maximum number of base stations with a processing load $L_{\max}$ that can be scheduled on a GPP platform. Conversely, we can determine the maximum feasible processing load $L_{\max}(M)$ for a given choice of M. Now we have a set of tuples for an N core platform

$$\{(M_1, L_{\max}(M_1)), (M_2, L_{\max}(M_2)), \ldots, (M_K, L_{\max}(M_K))\}, \tag{11}$$

where M_k is the maximum number of base stations that can be scheduled on a single GPP platform and $L_{\max}(M_k)$ is the corresponding maximum processing load.

Now consider a maximum processing load of $L_{\max}(M_k)$. We say a subset $\mathcal{J}$ of base stations $\{B_j\}_{j\in\mathcal{J}}$ is feasible for this load if the sum of the individual base station loads corresponding to a statistical guarantee P_F is less than $L_{\max}(M_k)$, i.e., $\sum_{j\in\mathcal{J}} L_{j,t}(P_F) \;\leq\; L_{\max}(M_k)$. These base stations can be combined together to create a *super* base station whose processing load is bounded by $L_{\max}(M_k)$. Now, we can schedule M_k super base stations on the GPP platform.

Our goal is to minimize the total number of platforms for scheduling the base stations. Instead, we partition the base stations into the minimum number of subsets, where each subset can be viewed as a super base station corresponding to a processing load $L_{\max}(M_k)$ for some k. Then, we group M_k of the super base stations corresponding to processing load $L_{\max}(M_k)$ into a single platform. We perform cyclic scheduling over the super base stations in order to process their subframes. If a super base station is scheduled for processing, then the subframes of the individual base stations constituting the super base station are processed in order. If the processing time of the subframe of base station B_j exceeds the allocated time $L_{j,t}(P_F)$, then the subframe is dropped from the queue. This approach represents the separation principle and the delay guarantees for each base station is met by virtue of the cyclic scheduling framework.

To compute the minimum number of super base stations, we note that each choice of processing load $L_{\max}(M_k)$ for some k can be viewed as a bin of size $L_{\max}(M_k)$. Hence, we have a variable size bin packing problem where the set of base stations with loads $\{L_{i,t}(P_F)\}_i$ have to be filled into minimum number of bins, where the size of the bins is from the set $\{L_{\max}(M_k)\}_{k=1}^{K}$. For certain mild assumptions satisfied here, the iterative first fit decreasing algorithm guarantees a $\frac{3}{2}$ approximation to the bin packing problem [17]. Since we group M_k of the bins (or super base stations) corresponding to $L_{\max}(M_k)$ into a platform, we might further step away from optimality if the number of bins of size $L_{\max}(M_k)$ is not a multiple of M_k. Firstly, this loss is bounded by K, which is the number of choices for bin size. Secondly, this loss can be further reduced by carrying out a second round of optimization on the platforms that are not fully occupied.

4.2 Simulation results

In Section 3, we evaluated the possible gains in resource pooling by analysing real-world data. We use the same data to evaluate the performance of our architecture. We consider two scenarios. First, we study the gains from resource pooling at each cell site by calculating the total number of compute resources required to provide a certain statistical guarantee. We compare this against the base case where each cell site is provisioned to ensure that all base stations with peak load are processed successfully. The second case is when the resource pooling for all base stations is done at a central location for a certain statistical guarantee. We compare this against the case where the central site is provisioned to process all the base stations at peak load. We study both these scenarios for the busy hours of the day which range from 0800 hours to 2200 hours. Clearly the gains of resource pooling will be higher in the second case than in the first case. But this comparison will allow cellular operators to analyse the cost-benefit trade-off of resource pooling at a central cloud versus performing it at the cell sites.

We report results based on provisioning for a five core platform. The choice of a five core platform is motivated by the Intel processor on which we implement our design and this will explained in detail in Section 5. The five cores can be provisioned to handle $2, 3, 4, 5$ or 6 base stations[5]. The corresponding $L_{\max}$ for each of these choices is given by $5/M_i$, i.e., if we provision 4 base stations, then $L_{\max}(4) = 5/4 = 1.25$. We assume that each base station has a peak load of $5/3 = 1.66$ ms, which means that for providing hard guarantees, we can provision 3 base stations in a single platform. We made this choice for two reasons: firstly, three base stations correspond to three sectors in a cell site. Secondly, we ensure that the utilization of all the cores in the platform is 100% for the base case with peak load. This gives us conservative estimates of the pooling gains from bin packing for the case with statistical guarantees.

In this setup, for the base case of scheduling 175 base stations at peak load, we require 63 platforms at the cell site and 59 platforms at the central location. Note that more compute resources are required at the cell site since one of the platforms at a cell site can be over provisioned, i.e., it processes fewer than the number of base stations it is provisioned for. The results for resource pooling gains under statistical guar-

[5]We limit the maximum number of base stations that can be scheduled on a compute resource since we do not factor all the overheads associated with scheduling multiple base stations like limits of the network interfaces which might constrain our options.

Table 2: Reduction in load at the cell site vs. central location. The baseline for comparison is 59 platforms for central location and 63 for the cell site.

Prob. of failure (P_F)	Number of Proc. at Cell Site (%Savings)	Number of Proc. at Central Location (% Savings)
10^{-4}	45.39 (27.95%)	44.51 (24.55%)
10^{-6}	49.03 (22.17%)	46.86 (20.57%)
10^{-8}	52.41 (16.8%)	47.49 (19.5%)

antees are reported in table 2. We obtain them by computing the number of processors required in each 15 minute interval and average the results. We observe from table 2 that most of the gains under resource pooling are obtained at the cell site itself. However, there might be other benefits to perform resource pooling at a central location like substantial savings in operations and management and capacity gains via advanced signal processing techniques involving cooperative transmissions/receptions from neighboring cell sites. Further we plot the number of total processors required as a function of time for a statistical probability of failure of 10^{-8} in figure 6.

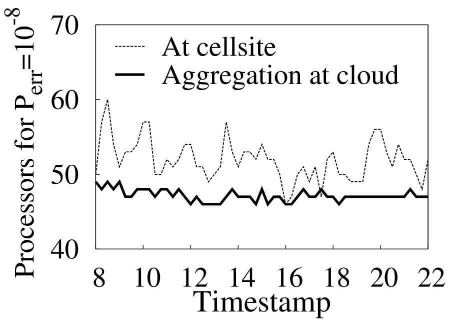


Figure 6: Number of processors required for aggregation at cell site or cloud at 15 minute intervals.

5. SYSTEM DESIGN

In this section we summarize our design exercise for implementing a multi-threaded LTE base-station. As mentioned earlier in Section 1, the basic building blocks of an LTE base station in our framework are ported from OpenAirInterface project [5]. The OpenAir project has developed a 3GPP compliant 5 MHz LTE implementation.The implementation executes each subframe in a single threaded fashion, i.e., it does not separate the user processing onto different threads as mentioned earlier. Furthermore it also runs the entire base station as a single thread and does not execute different subframes on different cores.

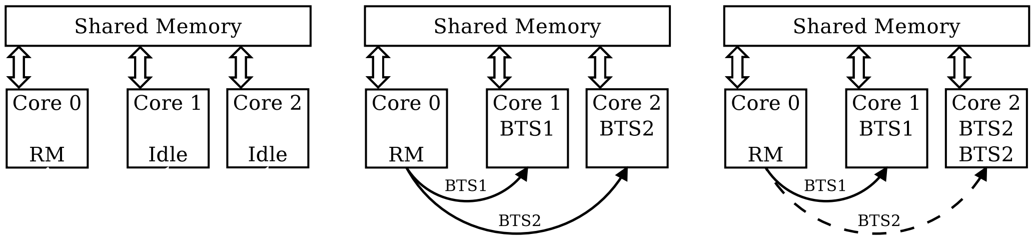
We implement our solution on an Intel Xeon W3690, with 6 physical cores and each core running at 3.47 GHz, 12 MB shared L3 cache and 256 KB L2 cache. Our platform uses Linux kernel 2.6.31 [9] with RT preemption patch (PREEMPT_RT) [4] whose default scheduling mechanism is Round-Robin [12]. But for better time predictability the threads can also be launched with FIFO scheduling option [12] and same *real-time priority* to prevent any preemption. We now describe some of the implementation challenges we faced and some design choices we made to overcome these issues.

Resident threads on cores: As explained in Section 4, our resource management framework dispatches different subframes of a single base station to different cores. Furthermore since LTE subframes arrive every 1 ms and the processing time for a subframe can exceed 1 ms, a base station can be concurrently processing multiple subframes. This implies that we require a multi threaded implementation for each base station that ensures that subframes can be processed concurrently without any conflicts. The central idea is to leverage the thread level parallelism (i.e. task parallelism) on a multicore general purpose processing platform to manage parallel processing modules of an LTE base station. There can be two approaches to this problem. First approach is to create a new thread for each subframe that has to be processed and assign this thread to a core according to the resource management algorithm (see Section 4. Although simple in design, this framework of transient threads has stability issues due to constraints of the underlying Linux kernel. We verified through experiments that creation and deletion of multiple threads at timescales of milliseconds overloads the system within a short span of 100 milliseconds. This is because the kernel is not able to free up resources at such timescales. Hence, we adopt the second approach of making each thread resident to a processor core and notifying the thread dynamically when a subframe needs to be processed.

Figure 7 explains our framework by showing how subframe processing jobs are launched in our system. For simplicity, we show only three cores out of which core 0 is always running the resource management algorithm (denoted by RM) and core 1 and 2 are dedicated for running LTE subframe processing. We assume that we are managing only two base-stations i.e. the RM process will launch two subframe processing jobs every millisecond. We also assume that with current processing loads, subframe processing for the first base-station (denoted by $BTS1$) will finish within a millisecond but the subframe processing for second base-station (denoted by $BTS2$) will not. Figure 7(a) shows that before time instance ($t = 0$) the threads running on core 1 and 2 are idle. In Figure 7(b) at time instance ($t = 0$), RM launches 2 subframe processing jobs to each of these core (denoted by solid arrow) and signals the idle threads to resume processing. In Figure 7(c) at time instance ($t = 1$), core 1 has finished the subframe processing of $BTS1$ but core 2 is still running the subframe processing of $BTS2$ and RM launches the next set of subframe processing jobs to these cores. At core 1, the new job is started immediately but at core 2 the job is enqueued (denoted by dashed arrow) after the previous job. Every resident thread maintains a *job-queue* where new jobs are enqueued by the RM. If the queue gets empty, the corresponding thread goes to sleep and the RM needs to wake-up the thread when a new job is dispatched to this thread.

FIFO order execution: Recall that the default scheduling mechanism of our Linux kernel is Round-Robin on each core. If two or more tasks are dispatched to a core and they are scheduled in Round-Robin fashion, it is difficult to predict their turn around times even if the individual processing loads are known. Therefore, we decided to incorporate FIFO order execution of multiple tasks on the same core. As mentioned earlier, this FIFO scheduling is implemented using an *job-queue* for each thread where new tasks are enqueued by RM. The thread goes to idle state if the queue is empty and needs to be signaled by RM after a new job is enqueued.

Circular shared buffers: To avoid conflicts between concurrent running processes, several shared buffers in OpenAir need to be duplicated (e.g. the IQ sample buffer for base station Rx processing modules). The degree of duplication however can be limited by the maximum number of concurrent



(a) Before Time $t = 0$ (b) At Time $t = 0$ (c) At Time $t = 1$

Figure 7: Resident Thread Framework and Signaling

Table 3: Average uplink subframe processing times. Single BTS and Dual BTS modes refer to one and two base stations scheduled on the processor respectively. Super base station mode refers to two base stations scheduled as a super base station. In Super BTS mode, processing time is the total time to process both the subframes.

Configuration		TBS	Uplink proc. time (ms)		
PRB	MCS	(Bytes)	Single BTS	Dual BTS	Super BTS
12	4	680	0.701	0.700	1.383
12	8	1480	0.794	0.788	1.557
12	12	2408	0.989	0.999	1.966
16	4	904	0.793	0.793	1.568
16	8	1928	0.899	0.911	1.786
16	12	3240	1.209	1.209	2.409
20	4	1160	0.883	0.885	1.756
20	8	2472	1.022	1.038	2.051
20	12	4008	1.526	1.525	3.035

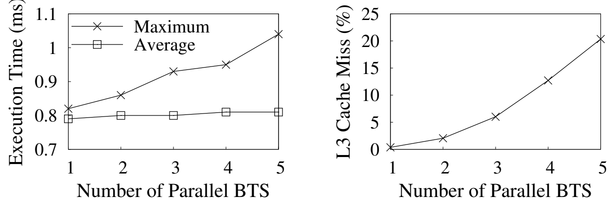
threads in our system. We maintain a circular list of shared buffers and while launching a new task assign the next free block to it and advance in the queue in circular fashion. This helps us to keep the allocated memory to our system as low as possible.

6. EXPERIMENTS

We now present a subset of the experiments we conducted to evaluate our system. We study the effects of sharing the compute resources between multiple base stations and verify that there is negligible deterioration in average processing time of each base station. We also observe that the worst case processing time for a base station can increase if multiple base stations share the compute resource. This effect can be accounted for in the resource management algorithm.

The details of LTE are presented in Section 2 and Figure 2 shows the time taken to process uplink and downlink subframes under different configurations (allocated PRB and MCS). Since the uplink processing is more intensive than downlink (see Figure 2), we focus on the performance of the uplink (base station Rx) processing. All our experiments are done with a single UE per base station and a 2 × 1 MIMO configuration. Each UE gets connected to a base station and thereafter remains static and does not cause any hand-over or disconnection. After the connection is established, the UE always has a full buffer and all available PRBs in every subframe are used for data transfer. For simplicity and predictability, we override the CQI based MCS calculation in the scheduler and instead choose a pre-defined MCS.

Table 3 compares the uplink processing times for a single base station when either one or two base stations are scheduled on the processor. The total number of data bytes to be transmitted in a subframe is called the transport block size (TBS) and is specified by LTE as a function of the allocated PRB and MCS (as shown in Table 3). The average subframe processing times are calculated over 18000 LTE subframes. Note that the average processing time for a subframe remains almost unchanged irrespective of whether we schedule one or two base stations on the processor. Recall that for providing statistical guarantees, we can combine multiple base stations to form a super base station. Table 3 also includes the processing time for a super base station comprising of two base stations. This is roughly double than that of a single base station. Although, average processing times remain same



(a) Maximum and Average uplink processing time (b) Percentage of L3 Cache (shared) miss

Figure 8: Effect of shared cache misses

for multiple base stations running in parallel, we observed a steady increase in the maximum subframe processing time as we increase the number of base stations in the system. Figure 8(a) shows the average and maximum subframe (uplink) processing times for multiple base stations. All uplink subframes correspond to 16 PRBs and MCS 4. The average processing time stays almost constant around 0.8 ms but the maximum (worst-case) processing time observed among all active base stations increases from 0.81 ms to 1.04 ms as we increase the number of base stations running in the system simultaneously. We found by further investigations that increased number of shared cache misses are the primary reason behind this increase in maximum processing time. Figure 8(b) shows the percentage of L3 cache misses with increasing number of base stations running in parallel. Note that, with original OpenAir code this effect of shared cache misses on worst-case execution time of a base station was more severe and we were unable to schedule more than two base stations on 5 cores (same configuration). But with careful analysis of cache misses and restructuring of large data structures, we managed to avoid obvious cache-misses in few of the computation-heavy functions (e.g. turbo decoding) which enabled us to schedule as many as 5 base stations. Making all the data structures in OpenAir code cache-conscious [10] or tuning each of the functions to avoid cache misses [13, 14] is beyond the scope of this work but we strongly believe that such exercises will further decrease the gap between average and maximum processing time of multiple base stations.

7. RELATED WORK

We briefly review the literature in the area of real-time systems. Most of the literature in traditional real-time systems considers tasks with an implicit deadline equal to their period [19]. Furthermore, tasks are assigned priorities for completion. For the case of scheduling such tasks on a single processing core (uniprocessor), the Earliest Deadline First (EDF) scheduler is known to be optimal [11]. On multi-core (or multiprocessor) systems, however, more complex schedulers such as the Proportionally Fair scheduler are needed for optimal scheduling [15]. In the tasks of interest to us, the deadline for completion can exceed the time-period. Such systems are called arbitrary-deadline cases and have been studied in the literature [7] [6] [18]. We are interested in scheduling such tasks with arbitrary deadlines on a multi-core system, while ensuring the separation principle. To the best of our knowledge, there are no algorithms in existing real-time literature that satisfy our requirements.

The cellular industry is currently evaluating proposals to transition towards a centralized architecture for baseband processing of the signals, see [2, 3]. Similar ideas for a centralized compute resource were evaluated for distributed antenna systems (DAS) [20], though DAS considered the design of large cells with distributed antennas to provide coverage.

8. REFLECTIONS

This paper is the first attempt at providing a rigorous resource management framework that allows a cellular operator to trade-off between the quaility of the network and the cost of operating the network (in terms of either the energy or the cost of baseband compute resources). However, there are several research issues that must be addressed to engineer such a system. Some of them are outlined below.

Reducing Backhaul Costs: The cost of fiber to backhaul the IQ samples to a central location can be prohibitive. For example, a 20 MHz LTE system requires around 1 Gbps of bandwidth per antenna to backhaul the samples. Hence, 3 sectors with each having a 4×4 MIMO system might require up to 12 Gbps of bandwidth. The cost of laying fiber with such high capacity can be prohibitive. As an alternative, the industry is considering architectures where some of the PHY processing is done at the base station, while the user specific samples are transported to the central site for processing. This has the potential to significantly reduce the backhaul requirements, while still permitting advanced signal processing techniques like CoMP and availing other benefits of the CloudIQ architecture.

Network and Switching Backplane at Data Center: We have primarily addressed the question of how to pool compute resources to minimize the number of required compute resources. We have not addressed the issues of network backhaul and the topology of the data center.

Resource Pooling Time Scales: In this paper, we have naively assumed that the resource pooling time scale is of the order of several minutes. Further we have assumed that the processing load distributions are predictable. An alternate approach would be to see if these decisions can be made at finer time scales and understand the predictability of the processing load distributions at those time scales.

Heterogeneous Systems: We have designed a resource management framework for homogeneous systems. However, currently deployed systems use heterogeneous components like accelerators for compute intensive functions like turbo decoding and GPP's for less intensive signal processing functions. As part of the ongoing work, we are extending the CloudIQ framework to heterogeneous systems.

Acknowledgements

We thank Laurent Thomas and Laurent Roullet for help in setting up OpenAir, and Martin Havemann, Kazukata Murakami, Francis Dominique and Jonathan Segel for useful comments. We would also like to thank the Wireless Business Division, Alcatel-Lucent, for help in obtaining the data.

9. REFERENCES

[1] 3GPP TS 36.211: LTE physical channels and modulation (Release 10).

[2] C-RAN: The road towards green RAN, China Mobile Research Institute. http://labs.chinamobile.com/article_download.php?id=104035.

[3] *http://http://www.ngmn.org.*

[4] *https://rt.wiki.kernel.org/.*

[5] *http://www.openairinterface.org/overview/acquiring.en.htm.*

[6] Andersson, B., Bletsas, K., and Baruah, S. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Real-Time Systems Symposium* (2008), IEEE, pp. 385–394.

[7] Baruah, S., and Fisher, N. Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. *Distributed Computing and Networking* (2008), 215–226.

[8] Berrou, C., and Glavieux, A. Near optimum error correcting coding and decoding: Turbo-codes. *Communications, IEEE Transactions on 44*, 10 (1996), 1261–1271.

[9] Bovet, D., and Cesati, M. *Understanding the Linux Kernel, Second Edition*, 2 ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[10] Chilimbi, T. M., Davidson, B., and Larus, J. R. Cache-conscious structure definition. In *Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation* (New York, NY, USA, 1999), PLDI '99, ACM, pp. 13–24.

[11] Dechter, R. *Constraint processing.* Morgan Kaufmann, 2003.

[12] Drozdowski, M. *Scheduling for Parallel Processing.* Computer communications and networks. Springer, 2009.

[13] Frigo, M., and Johnson, S. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on* (1998), vol. 3, IEEE, pp. 1381–1384.

[14] Frigo, M., Leiserson, C., Prokop, H., and Ramachandran, S. Cache-oblivious algorithms. In *Foundations of Computer Science, 40th Annual Symposium on* (1999), pp. 285–297.

[15] Garey, M., and Johnson, D. *Computers and intractability*, vol. 174. Freeman San Francisco, CA, 1979.

[16] Gesbert, D., Hanly, S., Huang, H., Shitz, S. S., Simeone, O., and Yu, W. Multi-cell mimo cooperative networks: A new look at interference. In *IEEE Journal on Selected Areas in Communications* (December 2010).

[17] Kang, J., and Park, S. Algorithms for the variable sized bin packing problem. In *European Journal of Operations Research* (2003).

[18] Lehoczky, J. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th* (1990), IEEE, pp. 201–209.

[19] Leung, J. *Handbook of scheduling: algorithms, models, and performance analysis*, vol. 1. CRC Press, 2004.

[20] Saleh, A., Rustako, A., and Roman, R. Distributed antennas for indoor radio communications. *Communications, IEEE Transactions on 35*, 12 (1987), 1245–1251.

[21] Tan, K., Zhang, J., Fang, J., Liu, H., Ye, Y., Wang, S., Zhang, Y., Wu, H., Wang, W., and Voelker, G. M. Sora: high performance software radio using general purpose multi-core processors. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (Berkeley, CA, USA, 2009), NSDI'09, USENIX Association, pp. 75–90.