Architectural and Cost Implications of the 5G Edge NFV Systems

Jianda Wang
Electrical and Computer Engineering Department
The University of Texas at Dallas
jxw174930@utdallas.edu

Yang Hu
Electrical and Computer Engineering Department
The University of Texas at Dallas
Yang.Hu4@utdallas.edu

Abstract

The recent issued 5G standard promises a comprehensive service provisioning for diverse user demands. The new challenges posed by 5G (e.g. the ultra-low latency communication) urge the functionality deployment to offload from network core to edge. Besides, the various requirements of QoS guarantee cast by fast-varying services make the Network Function Virtualization (NFV) an essential enabler of edge network evolution. To date, the performance challenges at the newly created NFV-enabled virtual network edge are still unexplored. In this paper, we investigate the converged virtual Radio Access Network (RAN) and Mobile Edge Computing (MEC) in 5G era. We first outline the solo virtual RAN performance. We then characterize the collocate workloads (RAN and MEC, RAN and RAN) on the commercial off-the-shelf (COTS) edge platform. Finally, we provide brief cost comparison between traditional RAN and virtual RAN. The implications of the virtual RAN system will benefit future edge NFV design.

1. Introduction

5G is a new paradigm that enables tremendous opportunities by delivering high-bandwidth and low-latency. The emerging 5G-enhanced applications include smart home, smart factory, Artificial Intelligence (AI), Augmented Reality (AR)/ Virtual Reality (VR) and autonomous driving, etc.

In responding to the fast-varying user service requirements and highly mixed traffics, the 5G network exploits Network Function Virtualization (NFV)[1] and network slicing technique to enhance its functional and architectural viability. NFV is a novel paradigm that enables scalable and flexible deployment of network services on edge or cloud infrastructure. The NFV-based network slicing technique accommodates various applications with unique service slices that across core network and edge network. However, configuring service slices at the network edge will create new research challenges for NFV.

The NFV at the network edge introduces new virtualization scenario and functions compared to the traditional NFV scenario. A trending edge NFV scenario is the virtualized Radio Access Network (vRAN). The RAN system is the most expensive part of the mobile network and the resource of 80% of performance problems that affect the user experience. The 5G RAN infrastructure calls for a re-architected service hierarchy to deliver a more flexible and diverse service provisioning. Compared to traditional LTE RAN, parts of the 5G core functions (i.e. user plane functions in the LTE core) and the baseband units (BBUs) are consolidated as 5G distributed units (DUs), where the non-real-time functions are implemented as virtual network functions (VNFs) or containers and deployed on commodity-off-the-shelf (COTS) servers to provide a more scalable and cost-effective solution compared to traditional specialized equipment at the cell site. Moreover, the vRAN is usually co-located with multi-access edge computing (MEC) workloads on the edge servers, such as streaming 360-degree video processing. Such mixed-service oriented workload consolidation can significantly challenge the resource management of edge NFV servers.

The new edge workload characteristics and deployment manners will significantly impact the edge hardware platform design. As the edge platform needs to seek the tradeoff between the cross-platform compatibility and the extreme cost-efficiency. However, though the existing work explores the virtualized BBU system under different RAN system configurations and provides initial insights on system computational capacity [2]. More detailed architectural implications are still needed to better decide the hardware architecture design trade-off of 5G edge NFV servers.

In this paper, we build a test framework of containerized 5G RAN and MEC. We collect the architectural characteristics of key RAN components and MEC applications on the 5G edge cloud platform. Our experiments demonstrate the following implications: (1) The RAN system is a CPU-intensive application, while its memory footprint is trivial. (2) The main performance bottleneck for RAN system is most Backend Bound. The optimization for Backend Bound is necessary to achieve better RAN performance. (3) In RAN system, turbo decoding module consumes majority part of CPU. The turbo decoding module should be highly optimized or it is suggested to implement by a hardware accelerator. (4) The co-running of RAN system and MEC application will slow down the MEC processing by 24% ~ 100%. However, the RAN system itself is not seriously affected when co-running with our chosen MEC applications. (5) The co-running of a RAN system with other RAN system will cause interference for each other. Even the occasionally resource confliction of RAN systems will destroy the RAN applications. It is suggested to binding multiple RANs to separated cores inside a server. (6) Compared to the current 4G RAN, the utilization of virtualized RAN can decrease the expenditure by 30% ~ 80% for the same band-

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00086

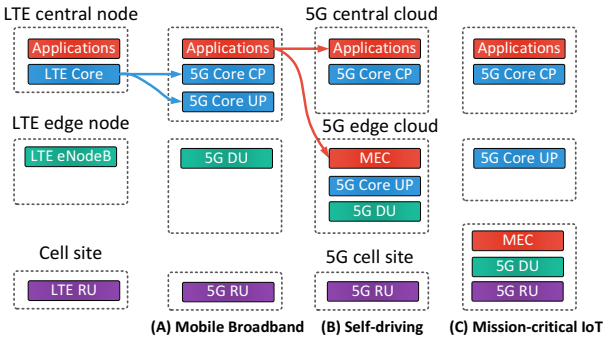


Figure 1. RAN architecture revolution from LTE to 5G.

width supporting. Considering utilizing MEC in access cell site, the decreasing percentage will be around 60% ~ 75%.

2. Background

To meet the rigorous requirements of bandwidth and latency, 5G needs a new network architecture that scales to device and traffic densities far beyond current LTE networks. As the most performance-critical part in the transport network, the traditional LTE radio access network (RAN) will be re-architected by combining parts of the core functions and edge computational capabilities. This will transfer the centralized RAN to a heterogeneous edge cloud. According to various service requirements of applications, the cloud-native function modules could be flexibly spawned and deployed on container-enabled central cloud, edge cloud, and cell site.

We show three typical cases of network function placement corresponding to the specific applications in Figure 1. For the mobile broadband service slices with round-trip delay tolerance is around 10ms, the 5G core control-plane functions and user-plane functions are collocated at the central cloud, while the 5G distributed unit (DU, the analogy to the eNodeB in LTE) is deployed on edge cloud servers. In the autonomous driving case with the round-trip latency guarantee is within 5ms, the 5G core data-plane functions will be deployed in edge cloud. And the edge cloud servers also host MEC platform to process latency-sensitive applications. For the industrial mission-critical IoT applications (e.g. robot motion control), the ultra-low-latency demand (<1ms) may even require host MEC on cell site.

3. Characterization Methodology

In this section, we first introduce our containerized edge cloud testbed, which is based on open-source virtualized RAN framework OpenAirInterface (OAI) and MEC Benchmarks. We then describe our experimental setup.

3.1 Experimental Platform Overview

We choose the OpenAirInterface (OAI) [3] as the RAN framework and conduct necessary function split to mimic a real 5G RAN system. OAI is the most complete open-source RAN experimentation and prototyping platform created by EURECOM. The OAI platform includes a full software

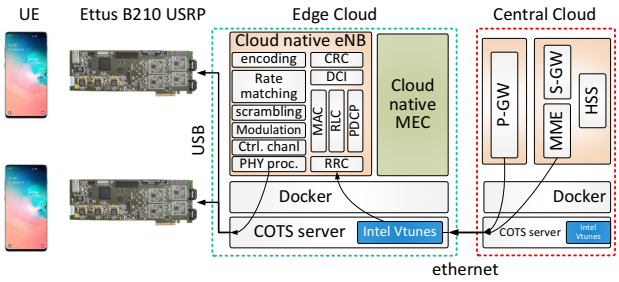


Figure 2. Typical RAN architecture for the Downlink Path

implementation of mobile cellular systems compliant with 3GPP standards in C under realtime Linux optimized for x86. For the 3GPP Access-Stratum, OAI provides standard-compliant implementations of PHY, MAC, RLC, PDCP and RRC, spanning the entire protocol stack from the physical to networking layer, for both eNodeB and UE. For the core network, the OAI provides standard-compliant implementations of a subset of 3GPP EPC components such as the Serving Gateway (S-GW), the Packet Data Network Gateway (P-GW), the Mobility Management Entity (MME), and the Home Subscriber Server (HSS). Fig. 2 shows a typical downlink path and the key network functions in OAI edge and core networks, note that both eNodeB and core functions are hosted in containers.

We select video benchmarks from PARSEC [4] to mimic our video processing based MEC applications. PARSEC includes emerging applications in system applications which mimic large-scale multithreaded industrial programs. We host selected PARSEC applications as containers.

3.2 Experimental Setup

Hardware Platform As shown in Figure 2, the experimental testbed consists of one/two units of Commodity-Off-The-Shelf (COTS) UE, one unit of OAI eNodeB Remote Unit (RU), one unit of eNodeB Distributed Unit (DU) and one unit of EPC. We use two sets of Intel Core machines (Core i7-8700 @ 3.20GHz 16GB RAM, and W2195 @ 2.30GHz, 128GB RAM) for eNodeB DU and RU, Intel Xeon machine (E5405 @ 2.00GHz 4G RAM) for EPC and Huawei Honor 8 as our UE. The testbed is implemented with a real RF front-end (Ettus B210 USRP).

Software Platform The eNodeB version we use is branch 2018_w25. For EPC, we use the developer branch. The Operation system used for both machines is Ubuntu 16.04. All the experiments were conducted with the same eNodeB configuration, namely FDD with 5 MHz bandwidth in band 7. We use Intel VTune Amplifier [5] to profile architectural data of key network functions as illustrated in Figure 2. All the applications are implemented in a docker container environment. The docker version we use is 18.09.1.

4. Understanding the Architecture Implications of Edge RAN system

In this section, we characterize architectural information of the solo-run eNodeB Distributed Unit (DU). We first profile the CPU and Memory Usage for the overall Distributed Unit (DU). We then evaluate the DU's submodules' CPU utilization and their micro-architectural characteristics. We demonstrate several key learning that are drawn from our detailed architectural profiling. We reason about the interesting findings and provide a platform setup guideline for modern edge NFV workloads.

4.1 CPU and Memory Usage of eNodeB

Figure 3 and Figure 4 illustrate the CPU Utilization of the overall OpenAirInterface eNodeB in both uplink and downlink under 3 traffic types: (1) UDP - increasing bandwidth while maintaining the same packet size (using default packet size). (2) UDP – decreasing the packet size while maintaining the same bandwidth. (3) TCP – increasing bandwidth while maintaining the same packet size (using default packet size). We choose 10Mbps for downlink and 5Mbps for uplink. We observe that the CPU Utilization increases with the increase of UE's packet per second (pps). Meanwhile, the memory usage always stays around 0.928GB regardless of the trend of the UE's alters, which illustrates that the *eNodeB Distributed Unit is a computation-intensive application*. Besides, the uplink per Mbps CPU utilization is two times to the downlink per Mbps CPU utilization since the uplink decoding algorithm is much more complicated compared to the downlink encoding algorithm.

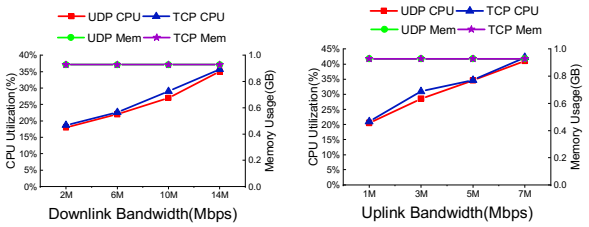


Figure 3. CPU/Memory Usage vs. UDP/TCP Bandwidth

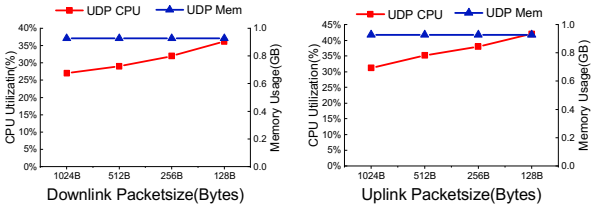


Figure 4. CPU/Memory Usage with UDP Packetsize

4.2 CPU Utilization for OAI eNodeB sub-modules

Figure 5, Figure 6 and Figure 7 show the CPU Utilization of sub-modules inside OpenAirInterface eNodeB for UDP traffics with different bandwidth/packet sizes and TCP traffics with different bandwidth. We observe that the layer 2 modules (PDCP, RLC, MAC) are less CPU-consuming compared to the physical layer modules (Encoding/Decoding, Rate Matching/Dematching, Scrambling/Descrambling and Modulation/Demodulation) in both downlink and uplink directions under all cases. *This indicates that the layer 2 modules are not constrained by CPU resource.* More attention needs to be paid into the physical

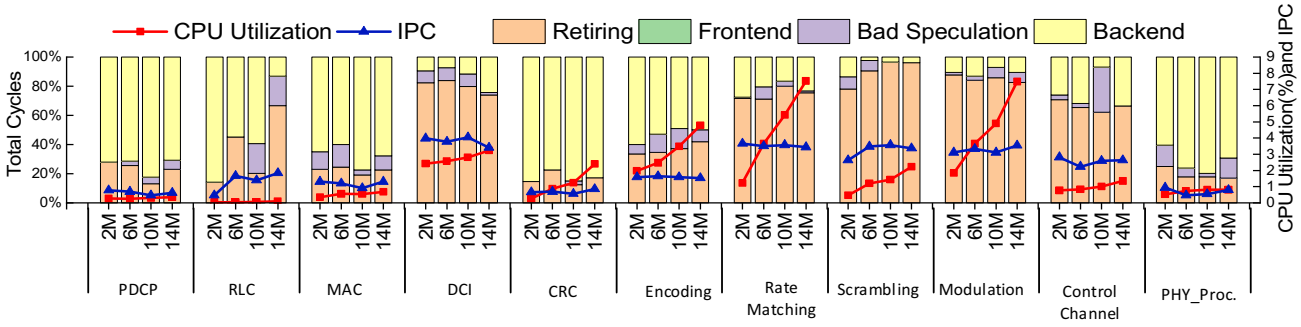


Figure 5a. CPU Utilization and Microarchitecture value for Downlink UDP Bandwidth

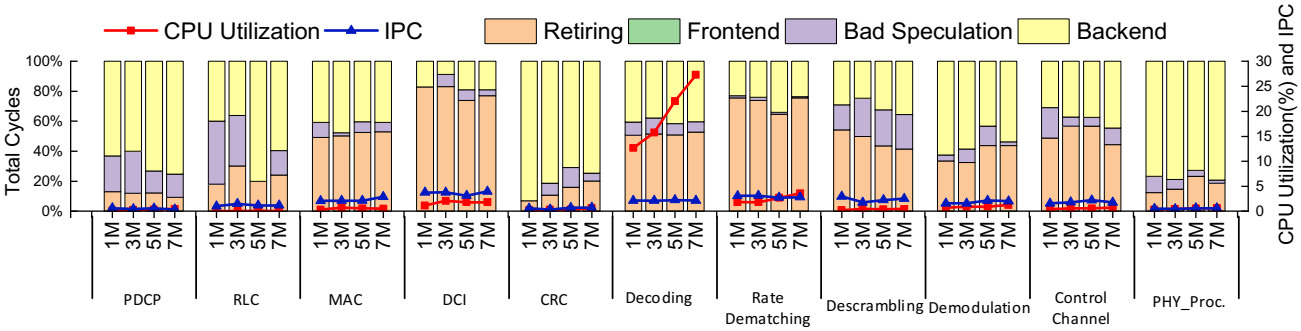


Figure 5b. CPU Utilization and Microarchitecture value for Uplink UDP Bandwidth

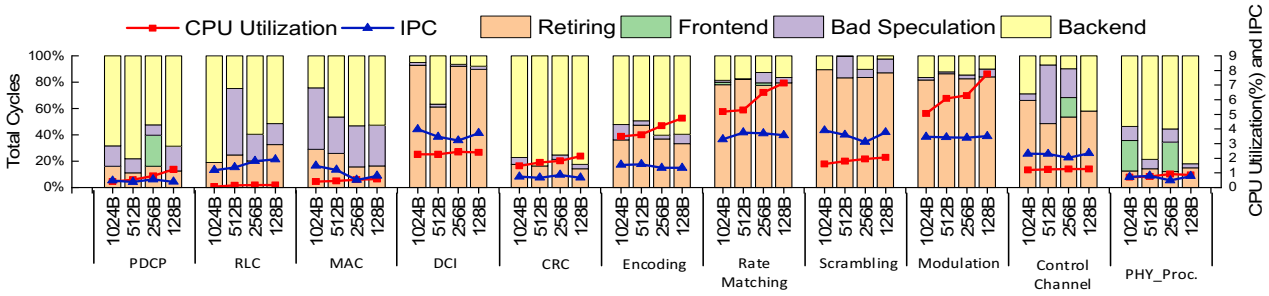


Figure 6a. CPU Utilization and Microarchitecture value for Downlink UDP Packetsize

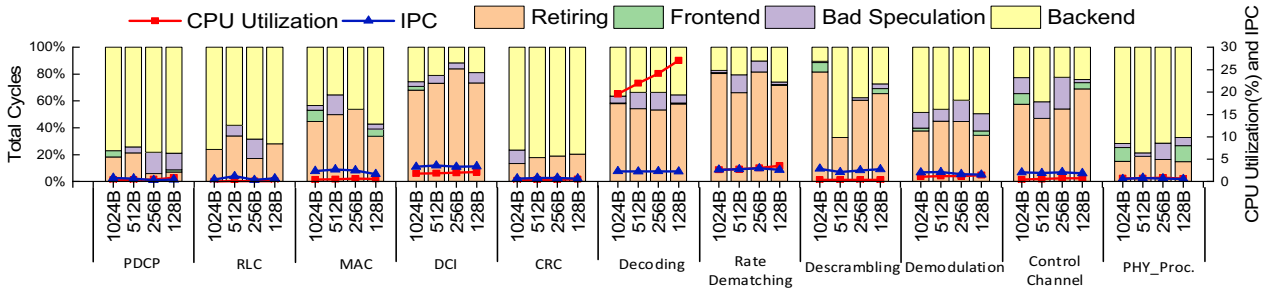


Figure 6b. CPU Utilization and Microarchitecture value for Uplink UDP Packetsize

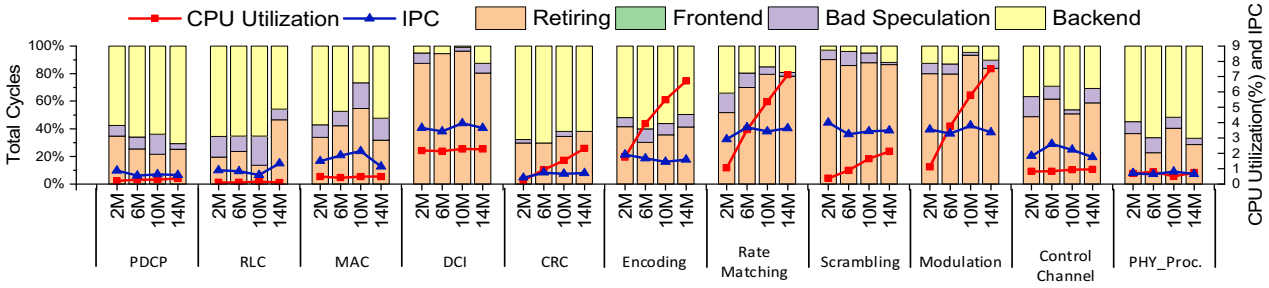


Figure 7a. CPU Utilization and Microarchitecture value for Downlink TCP Bandwidth

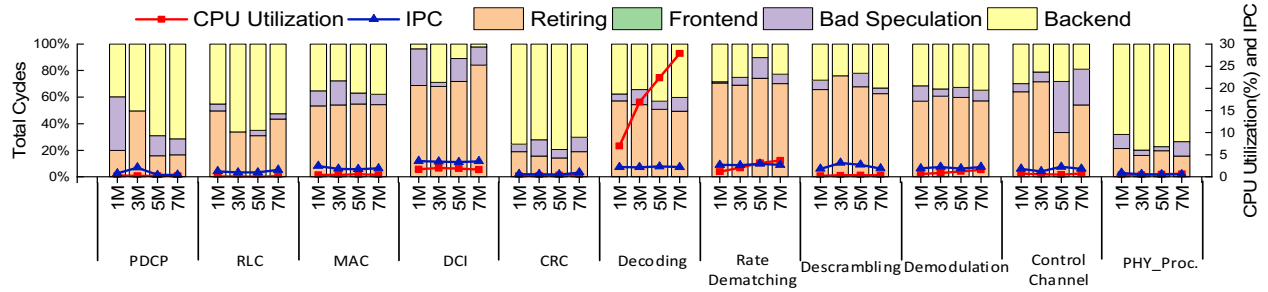


Figure 7b. CPU Utilization and Microarchitecture value for Uplink TCP Bandwidth

layer modules.

Figure 5, Figure 6 and Figure 7 also illustrate that the *CPU utilization of different modules inside eNodeB alter greatly in uplink and downlink direction.* For the downlink, the CPU consumptions are evenly distributed among the modules. For the Uplink, turbo decoding consumes the most of the CPU. The CPU utilization of all other modules is negligible compared to the turbo decoding. This result agrees with the studies on the Radio Access Network sub-module behaviors in [6]. The *Turbo decoding module should be highly optimized or this module is suggested to offloaded to a hardware accelerator.*

Besides, from Figure 5, Figure 6 and Figure 7 we observe that the CPU Utilization increases when the bandwidth goes up or the packet size goes down. The reason for this is that the total pps goes up when we increase the bandwidth or decrease the packet size in the experiment.

Table 1. Downlink Main Functions in Wimpy Node

Downlink Functions	Retiring	Memory Bound	Core Bound
sub_block_interleaving_turbo	72.9	13.6	14.9
threegpplte_turbo_encoder	30.4	30.5	34.3
crc24	15.2	48.5	36.4

Table 2. Uplink Main Functions in Wimpy Node

Uplink Functions	Retiring	Memory Bound	Core Bound
phy_threegpplte_turbo_decoder16	50.8	23.8	23.8
sub_block_deinterleaving	68	14.9	13
ulsch_decoding	38.1	50.7	8.5
_mm_sub_spi16	66.9	8	25.8
_mm_max_epi16	58.8	28.5	11.4
crc24	26.7	35.7	32.6

Moreover, we observe that the maximum bandwidth for TCP is slightly smaller compared to the UDP. Under Resource Block 25, the maximum bandwidth for TCP is around 7m for uplink and 15m for downlink. The maximum bandwidth for UDP is around 8m for uplink and 16m for downlink. The reason for this is probably that the TCP needs to send ACK which leads to smaller available maximum bandwidth. The trend for CPU consumption for each module of UDP and TCP are similar to each other.

4.3 Instructions Execution

Instruction per cycle (IPC) is a fundamental performance metric, which is used to measure instruction-level parallelism. There are four micro-architectural metrics relates to the IPC – Retiring, Bad Speculation, Frontend Bound and Backend Bound in Intel VTunes [7]. A high percentage of retiring usually means a high IPC value of an application. The high percentage of the other three categories will hurt the retiring, which will lead to low IPC. A thorough analysis of Frontend Bound, Bad Speculation and Backend Bound would help us locate the hotspot of an application and provide optimization direction for further development.

Figure 5, Figure 6 and Figure 7 show IPC values of the modules inside OAI eNodeB. For the downlink cases, we can see that the IPC value for modules DCI, Rate Matching, Scrambling and Modulation are near to 4, which is almost the ideal value. The IPC values for Control Channel are around 2.7, which is also acceptable. The Turbo Encoding module's IPC value is around 1.8, which suggests potential optimizations to improve the performance. For the uplink, we observe that the IPC values for all the modules are around 2, *which means the headroom still exists to get better performance of each module in eNodeB uplink direction.* Detailed analysis of microarchitecture bottleneck is given in section 4.4. Figure 5, Figure 6 and Figure 7 also provide the Retiring percentage of modules inside OAI eNodeB, we note that Retiring correlates well with IPC value, high retiring value always means high IPC.

4.4 Frontend Bound, Bad Speculation and Backend Bound Behavior

Figure 5, Figure 6 and Figure 7 also show the cycle breakdown for the submodules of DU. Frontend Bound denotes that instruction-fetch stall will prevent core from making forward progress due to lack of instructions. Bad Speculation reflects slots wasted due to incorrect speculations. Backend Bound illustrates that no uops are being delivered at the issue pipeline, due to lack of required resources in the Backend. We can see that across all the modules, *the Frontend Bound and Bad Speculation overheads are negligible*. The main stall of DU application is concentrated at Backend Bound, which means *the optimization for Backend part is necessary for DU application*. For the most CPU consuming module - turbo decoding, we observe that the Backend Bound is around 45%, which is the main reason causing the low IPC for the turbo decoding.

Table 4. Downlink Main Functions in Beefy Node

Downlink Functions	Retiring	Memory Bound	Core Bound
sub_block_interleaving_turbo	55.8	0	44.2
threegpplte_turbo_encoder	40	0	60.2
crc24	13.8	0	86.2

Table 5. Uplink Main Functions in Beefy Node

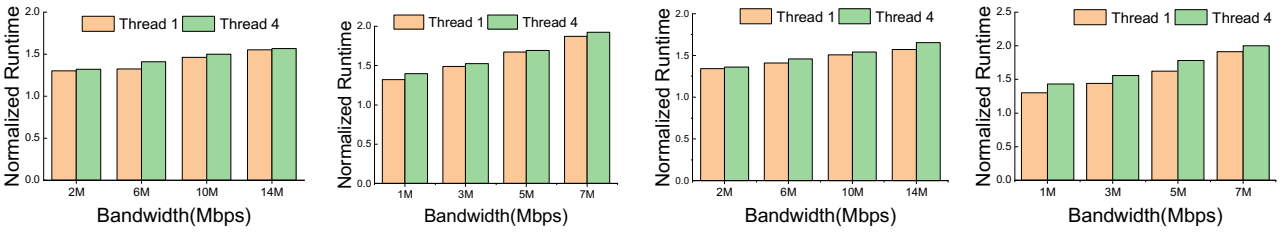
Uplink Functions	Retiring	Memory Bound	Core Bound
phy_threegpplte_turbo_decoder16	62.2	0	37.5
sub_block_deinterleaving	67.6	0	32.4
ulsch_decoding	45.7	0	52.8
_mm_sub_spi16	56.6	0	43.9
_mm_max_epi16	52.2	0	43.5
crc24	43.5	0	56.5

Table 3. Cache Size and Frequency in Wimpy and Beefy Node

	Wimpy Node	Beefy Node
L1 cache	384KB	1152KB
L2 cache	1536KB	18432KB
L3 cache	12288KB	25344KB
Frequency	3.2GHz	2.3GHz

We further investigate the source of Backend Bound by dividing it into two separate metrics: Memory Bound and Core Bound. Memory Bound manifests with execution units getting starved after a short while. Core Bound manifests either with short execution starvation periods or with sub-optimal execution ports utilization. Table 1 and Table 2 show CPU consumption dominant functions in the main modules. We can find the root cause of non-uniform Backend Bound. *Memory Bound and Core Bound both suffer from the current RAN system.* Memory Bound can be mitigated by simply increasing the cache size of the server, or by leveraging memory which can control the mapping of instruction and data storage for each core. Moreover, since the RAN system processing proceeds in continuous phases, providing buffers for data communication from one phase to another will probably mitigate the Memory Bound. Core Bound can be mitigated with the better code generation, e.g., avoiding dependent arithmetic operations in a sequence, or better vectorization organization of OpenAirInterface system. Furthermore, since the RAN system involves much on the complex number operations; perhaps leverage specific instruction function units for complex operations will promote the core performance for RAN system.

We further deploy the DU on alternative high-end COTS server platform *(W2195 @ 2.30GHz, 128GB RAM)* with higher cache size to investigate if the Backend bound overheads could be mitigated. Table 3 compares the cache size



(a) Downlink+vips (b) Uplink+vips (c) Downlink+x264 (d) Uplink+x264

Figure 8. The Influence of edge computing applications' runtime when co-running with eNodeB DU

and CPU frequency of the wimpy server and the beefy server.

Table 4 and Table 5 show the memory-bound and core bound details of CPU consumption dominant functions in eNodeB on the beefy server. We observe that the memory-bound is significantly mitigated because of the larger cache resources. However, the Core Bound overhead deteriorates on the beefy server. The reason for this is probably due to the frequency reduction of the server. The frequency reduction may lead to longer execution starvation and worse port utilization, which results in the higher Core Bound in beefy server environment. The counteracts of lower Memory Bound and higher Core Bound makes the overall Backend performance stay similar to the wimpy server platform.

5. Understanding the Architecture Implications of Co-location Edge systems

In this section, we start evaluating the performance of Distributed Unite (DU) and edge computing application when co-running them together on the same cores. We then leverage multiple vRANs on the same cores and show up the behavior.

5.1 Co-location of RAN and MEC Application

We begin with the evaluation of the performance for edge computing applications when co-running with eNodeB based DU. Furthermore, we demonstrate the performance of eNodeB when sharing the same cores with edge computing applications.

We select video processing related applications as our benchmarks since most of today's edge computing utilizations are concentrated on video processing. We choose vips and x264 from PARSEC. The workload information is shown in Table 6. Vips is an image processing library and x264 is an application for encoding video streams in the H.264 compression format. We first co-run edge computing applications and eNodeB DU with various traffic bandwidth on the same cores and report the edge computing applications' processing time. Figure 8 shows the normalized processing time of MEC applications. With the bandwidth going up for eNodeB, the edge computing applications require more time on completing the same-volume workload. The prolonged processing time is between 24% (when the bandwidth is 2M) to 100% (when the bandwidth is 14M). The reason for the longer processing time is that the eNodeB DU utilizes more CPU with the bandwidth increasing, which constraints the available CPU for edge computing application.

Table 6. Edge Computing Applications

	Information	Workload
vips	Image processing	Image 18K*18K pixels
x264	Video encoding	25fps 640*360 pixels

Furthermore, we evaluate the eNodeB Distribute Unit performance when it is co-running with edge computing applications. To our surprise, the performance of eNodeB does not degrade even though it is sharing the same core with the edge computing applications. Figure 9 shows typical performance for main modules inside eNodeB during the co-running, we find that the performance is almost the same compared with the solo eNodeB running situation (Figure 5). We observe that the PARSEC applications and eNodeB efficiently utilize the given cores. The utilization goes up to 100% for the leveraged cores. However, when we increase the bandwidth for the eNodeB, the CPU utilization of eNodeB goes up to the same level as the level without the co-running. The available CPU Utilization of PARSEC edge computing is constrained by eNodeB application during co-running.

Our speculation is that the CPU resource for eNodeB does not achieve its upper bound. All containers' default CPU-shares value is 1024, since the saturated traffic of eNodeB only consumes 45% of the CPU, it does not achieve its maximum available 50% CPU ratio when co-running with the edge computing application container.

In order to consolidate our speculation, we deploy 3 containers in an additional experiment. We create two containers for edge computing applications, with the CPU-share value 1024 and 256 respectively. We create a third container for eNodeB with the CPU-share value 128. Our expectation is that the CPU distribution ratio among the applications should be around 8:2:1. The CPU utilization for eNodeB should not exceed 20% when co-running with the other two applications.

However, the result of the experiment shows that the CPU utilization for eNodeB still cannot be constrained. For saturated traffic, the CPU Utilization of eNodeB goes up to 45%, which is the same value when we leverage the eNodeB alone in the given core. The CPU-share for the two edge computing application works as what we expect. The

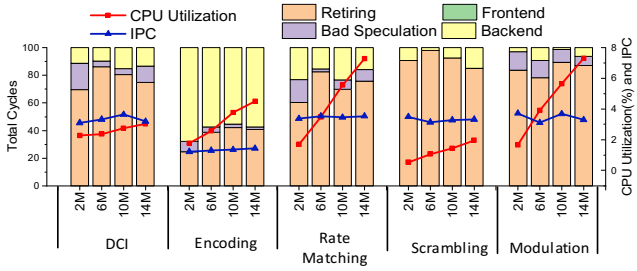


Figure 9a. CPU Utilization and Microarchitecture value for Downlink UDP with vips

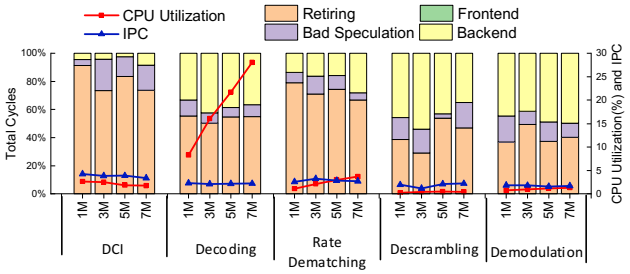


Figure 9b. CPU Utilization and Microarchitecture value for Uplink UDP with vips

CPU Utilization ratio for them is 4:1. Our further speculation is that there may exist inherited priority inside eNodeB and edge computing applications. The eNodeB's CPU Utilization should be constrained when co-running with another eNodeB since they have same level priority. We outline the performances of co-running eNodeB Distributed Unit under multiple cases in section 5.2.

5.2 Co-location of RAN and RAN Application

This section we illustrate the behavior of eNodeB Distributed Unit when it is co-running with other eNodeB Distributed Unit.

Figure 10 provides the experiment setup. Since we only have one USRP B210 board, we set OAI simulators as our co-runner with real USRP B210 board. The eNodeB is separated as Distributed Unit (DU) and Remote Unit (RU). The simulations' traffic is generated at RU end. In order to saturate the CPU, we create 2DUs with 2RUs under simulation mode as our co-runner.

We bind RAN (eNodeB DU) with a real board in core 1 of the server. We bind cores for OAI simulators under 3 cores sharing types: (1). The OAI simulators are bound to core 1, which is the same core with real board RAN. One core cannot sufficiently support simulators and real board RAN simultaneously. (2) The OAI simulators are bound to core 0 and 1. Under this situation, the simulator partially shares cores with real board RAN. The total cores are sufficient to support both OAI simulator and real RAN theoretically. (3) The OAI simulators are bound to core 0 and 2, in which case the simulators are completely separated with the real board RAN. The simulators and real RAN are provided with sufficient resources they required.

In case (1), we firstly start up real board RAN system and then we trigger the OAI simulators. The RAN system works smoothly when the simulators are not started up. When the simulators come in, the real board RAN is completely stuck by the simulators. We observe that the packet processing time violates the real-time bound inserted in the RAN system since the CPU resource is constrained. In case (2), since the simulations are bound to 2 cores, the OAI simulators utilize the cores dynamically. The total utilization of 2 cores for simulator stays the same but the utilization for each core is not stable, the CPU utilization swings between two cores. We firstly start the real board RAN and then we trigger OAI simulators. When the OAI simulators come in, we find that the OAI simulators and RAN system work properly if the CPU resources of two systems do not exhaust any of the cores. However, the RAN and OAI simulator will be stuck when any of CPU cores is exhausted. In case (3), the real board RAN and OAI simulators since both of the systems are provided with sufficient CPU resources.

From the experiment, we can conclude that (1) For real-time systems like RAN, sufficient resources are obligated. The system will be stuck if it cannot get its required resources. (2) For real-time applications, it is essential to set up applications in separate cores. The occasional resource deficiency under resources sharing policy will destroy the running applications (3) The real-time applications can co-run with the non-real-time applications such as vips and x264. The non-real-time applications do not compete resources with real-time applications. The co-running of real-time and non-real-time applications can promote resources utilization efficiency.

6. Edge Infrastructure Cost Analysis

This section we firstly provide a cost comparison between traditional RAN and virtualized RAN. Furthermore, we estimate the cost of the RAN system with MEC applications under traditional and virtualized RAN infrastructure.

Traditional RAN: Traditional RAN utilizes vendor-specific hardware device to support radio access network. To date, the maximum transmission rate one RAN can support is around 300Mbps [8], [9]. The cell site does not have to support the maximum traffic volume. Based on the traffic volume requirement of an area, the total traffic volume one RAN needs to support alters greatly, which causes the price variation for specific RAN. In this paper, we categorize the traffic volume as 4 levels: Sparse (50M), Normal (100M), Dense (200M) and Ultra Dense (300M). Since there is no open-source price for BBU equipment, we use approximate price for each RAN based on the report [10]. The price for the RAN under each level is shown in Figure 11. We assume the price of RAN in the report is for maximum traffic volume 300M and the price of RAN is proportional to its supported traffic volume. Figure 11 presents the price for traditional RAN.

Virtualized RAN: Virtualization is the trend for the RAN infrastructure. Typical server price can be found in the link [11]. From the experiments we observe that the CPU

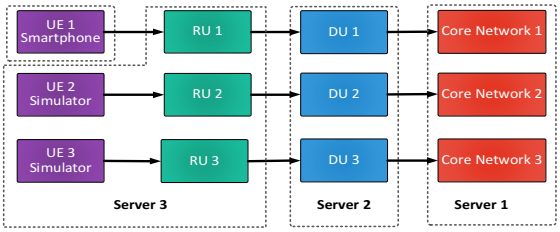
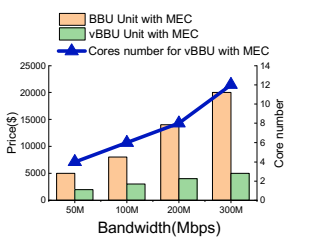


Figure 10. RAN co-running with RAN setup



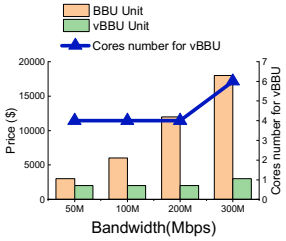


Figure 11. BBU&vBBU prices **Figure 12. BBU&vBBU MEC prices**

consumption for downlink is around 1.5% per Mbps. The required core number can be calculated accordingly, e.g., to support 300Mbps bandwidth, the total CPU utilization will be 450%, which means that a 6 cores server is recommended. We show up the vRAN price under the 4 levels traffic volume in Figure 11. From Figure 11, we observe that vRAN decrease the expenditure from 30% to 80% compared with the traditional RAN.

RAN with MEC: In order to support ultra-low latency applications and mitigate the traffic volume for the core network, MEC will be widely used in the near future. Traditional RAN requests the additional COTS server to support MEC applications while the vRAN can support the MEC applications with its extra resources inside COTS server. To the best of our knowledge, there is no open-source data on servers' average CPU utilization for MEC. The data we can find is Google's cloud CPU utilization estimation [12], the Google cloud's CPU utilization is between 30% ~ 50%. MEC workloads are not completely the same with the workloads in the data center today, there is the trend that utilizes MEC to implement traditional data center missions [13], [14]. For this reason, we leverage the data in [12] as MEC CPU utilization value. We assume that at most 50% of each core can be utilized by RAN workload. Besides, extra 20% CPU resource is provided as redundant. E.g., for bandwidth 300Mbps, 450% CPU utilization is required for RAN. We assume MEC workloads utilize 50% of each core so that 450% CPU utilization is required by MEC. The total is 900%. Given 20% redundant resource, the total CPU utilization of the server is 1080%. Thus a 12 core server is recommended to handle the RAN and MEC workloads. For traditional RAN, the total site expenditure is its BBU device price plus a 4 cores server price. Figure 12 presents the price comparison for traditional RAN and vRAN with MEC. We observe that the expenditure of vRAN with MEC is 60% ~ 75% lower compared with the traditional RAN.

7. Related Work

Edge NFV: Network Virtualization Networks (NFV) has received substantial attention from the research community in recent years with both academia and industry recognizing its benefits on operational mobile networks. Although most of the work highlights the NFV process on core networks [15], [16], [25], [17]–[24], there are still several NFV projects [3], [26]–[28] proposed at the edge network. While the scope of the above-mentioned work includes the RAN virtualization and network slicing realization, none of them provide a detailed architectural characterization of virtualized RAN system, particularly as cloud-native containerized form. Moreover, no existing work has explored the co-located vRAN and MEC system from an architectural perspective.

RAN Characterization: In the last few years, several works [29]–[34] provide the performance analysis and study on the vRAN system. [30], [31] introduce the concepts and architecture of the vRAN system. [30] validates two MAC schedulers and analyzes the vRAN system, in terms of memory occupancy and execution time. [31] performs thorough profiling of OAI, in terms of execution time, on the user plane data flow. A recent work [2] explores the system computational requirements of vBBU on a cloud RAN testbed. However, none of the existing work has provided a comprehensive architectural behavior characterization for vRAN framework. *To the best of knowledge, our work is the first one that explores the architectural implications of the co-location of cloud native vRAN and MEC.*

8. Acknowledgments

We thank all the anonymous reviewers for invaluable and insightful comments to make this paper better. This work is supported in part by NSF grant CCF-1822985. The corresponding author is Yang Hu.

9. Conclusion

In this paper, a comprehensive workload characterization is presented for virtual RAN and edge computing. Based on our characterization work, the main bottleneck for the virtual RAN is its Backend Bound. Besides, when the RAN is co-running with the edge computing applications, the edge computing applications are slowed down by the RAN system. However, the performance of the RAN is not affected by our chosen edge computing applications. When the RAN is co-running with RAN, they will interfere with each other. Finally, we show up that virtual RAN will decrease the expenditure on both solo RAN build up and RAN with MEC build up compared with traditional RAN

References

[1] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," *Proc. 11th USENIX Symp. Networked*

Syst. Des. Implement. (NSDI 14), vol. 12, no. 1, pp. 445–458, 2014.

[2] T. X. Tran, A. Younis, and D. Pompili, "Understanding the Computational Requirements of Virtualized Baseband Units Using a Programmable Cloud Radio Access Network Testbed," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*, 2017, pp. 221–226.

[3] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *ACM SIGCOMM Comput. Commun. Rev.*, 2014.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 72–81.

[5] "Intel® VTune™ Amplifier XE," in *Power and Performance in Enterprise Systems*, 2015.

[6] V. Venkataramani, A. Kulkarni, T. Mitra, and L.-S. Peh, "SPECTRUM: a software defined predictable many-core architecture for LTE baseband processing," 2019.

[7] A. Yasin, "A Top-Down method for performance analysis and counters architecture," in *ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

[8] K. Watanabe and M. Machida, "Outdoor LTE infrastructure equipment (eNodeB)," *Fujitsu Sci. Tech. J.*, 2012.

[9] S. M. Sonia Rathi, Nisha Malik, Nidhi Chahal, "Throughput for TDD and FDD 4 G LTE Systems," *Int. J. Innov. Technol. Explor. Eng.*, 2014.

[10] "https://www.fiercewireless.com/wireless/mavenir-wants-to-replace-proprietary-baseband-unit-x86-and-software."

[11] "https://www.dell.com/en-us/work/shop/workstations-isv-certified-dell/sf/precision-desktops?appliedRefinements=14602."

[12] P. Garraghan, P. Townend, and J. Xu, "An analysis of the server characteristics and resource utilization in Google cloud," in *Proceedings of the IEEE International Conference on Cloud Engineering, IC2E 2013*, 2013.

[13] Q. Cui *et al.*, "Stochastic Online Learning for Mobile Edge Computing: Learning from Changes," *IEEE Commun. Mag.*, 2019.

[14] J. W. Chengcheng Zhao, Mianxiong Dong, Kaoru Ota, JianHua Li, "Edge-MapReduce-Based Intelligent Information-Centric IoV: Cognitive Route Planning," *IEEE Access*, 2019.

[15] C. Guo *et al.*, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International Conference on Emerging Networking Experiments and Technologies, Co-NEXT'10*, 2010.

[16] T. Koponen *et al.*, "Network Virtualization in Multi-tenant Datacenters," *Proc. 11th USENIX Symp. Networked Syst. Des. Implement. (NSDI 14)*, pp. 203–216, 2014.

[17] M. R. Sama, X. An, Q. Wei, and S. Beker, "Reshaping the Mobile core network via function decomposition and network slicing for the 5G era," in *2016 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2016*, 2016.

[18] R. Ravindran, A. Chakraborti, S. O. Amin, A. Azgin, and G. Wang, "5G-ICN: Delivering ICN Services over 5G Using Network Slicing," *IEEE Commun. Mag.*, 2017.

[19] M. Bagaa, T. Taleb, A. Laghrissi, A. Ksentini, and H. Flinck, "Coalitional game for the creation of efficient virtual core network slices in 5G mobile systems," in *IEEE Journal on Selected Areas in Communications*, 2018.

[20] B. Chatras, U. S. Tsang Kwong, and N. Bihannic, "NFV enabling network slicing for 5G," in *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*, 2017.

[21] Y. Hu, M. Song, and T. Li, "Towards 'Full Containerization' in Containerized Network Function Virtualization," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 467–481.

[22] Y. Hu and T. Li, "Towards Efficient Server Architecture for Virtualized Network Function Deployment: Implications and Implementations," in *Proceedings of the 49th International Symposium on Microarchitecture - MICRO-49*, 2016.

[23] Y. Hu and T. Li, "Enabling Efficient Network Service Function Chain Deployment on Heterogeneous Server Platform," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[24] S. G. Kulkarni *et al.*, "Nfvnice: Dynamic backpressure and scheduling for NFV service chains," in *SIGCOMM 2017 - Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication*, 2017.

[25] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An Analysis and Empirical Study of Container Networks," in *Proceedings - IEEE INFOCOM*, 2018.

[26] A. Virdis, G. Stea, and G. Nardini, "SimuLTE - A modular system-level simulator for LTE/LTE-A networks based on OMNeT+," in *SIMULTECH 2014 - Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, 2014.

[27] W. Z. Qinghua Zheng, Haipeng Du, Junke Li, "Open-LTE: An Open LTE Simulator For Mobile Video Streaming," *2014 IEEE Int. Conf. Multimed. Expo Work.*, 2014.

[28] L. Zhang *et al.*, "Characterizing and Orchestrating NFV-ready Servers for Efficient Edge Data Processing," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 22:1--22:10.

[29] A. Virdis, N. Iardella, G. Stea, and D. Sabella, "Performance Analysis of OpenAirInterface System Emulation," in *Proceedings - 2015 International Conference on Future Internet of Things and Cloud, FiCloud 2015 and 2015 International Conference on Open and Big Data, OBD 2015*, 2015.

[30] C. Y. Yeoh, M. H. Mokhtar, A. A. A. Rahman, and A. K. Samingan, “Performance study of LTE experimental testbed using OpenAirInterface,” in *International Conference on Advanced Communication Technology, ICACT*, 2016.

[31] Po-Chiang Lin and Sheng-Lun Huang, “Performance Profiling of Cloud Radio Access Networks using OpenAirInterface,” *Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)*, 2019.

[32] Q. Zheng *et al.*, “WiBench: An open source kernel suite for benchmarking wireless systems,” in *Proceedings - 2013 IEEE International Symposium on Workload Characterization, IISWC 2013*, 2013.

[33] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, “srsLTE: An open-source platform for LTE evolution and experimentation,” in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2016.

[34] O. Neji, N. Chendeb, O. Chabbouh, N. Agoulmine, and S. Ben Rejeb, “Experience deploying a 5G C-RAN virtualized experimental setup using OpenAirInterface,” in *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband, ICUWB 2017 - Proceedings*, 2018.