Characterizing and Understanding the Architectural Implications of Cloud-native Edge NFV Workloads

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

In responding to the fast-varying user service requirements and highly mixed traffics, the 5G network exploits Network Function Virtualization (NFV) and network slicing to enhance its functional and architectural viability. With the new service provisioning requirements posed by 5G (e.g. the ultra-low latency and massive machine-type communication) urge the functionality deployment to offload from the network core to edge, the various demands of QoS guarantee cast by the diverse services make the NFV an essential enabler of edge network evolution. However, the newly created performance challenges at the network edge are still unexplored. In this paper, we investigate the converged RAN and MEC architecture in the 5G era. Specifically, we characterize the collocated cloud-native workloads (RAN and MEC) on the COTS edge platform and provide the architectural implication which will benefit the future edge architecture design.

1. Introduction

The recently issued 5G standard promises a surge in network bandwidth and an explosion in the number of connected devices. This communication revolution will enable consolidated service provisioning of many killer applications in addition to traditional voice and data communication, such as AI, AR/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) and network slicing 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.

To date, most of the existing NFV research focuses on the core network to provide a better data plane and control plane performance. With the new service provisioning requirements posed by 5G (the ultra-low latency and massive machine-type communication) urge the functionalities offload from the network core to edge, the various demands of QoS guarantee cast by the diverse services make the NFV an essential enabler of edge network evolution. The NFV-based network slicing technique enables unique service slices that are customized for various applications such as IoT, automated cars, streaming 360-degree videos, etc. *However, this will create new research challenges at the network edge*.

First, the NFV at the network edge involves a 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.

Second, the recent trend to adopt cloud-native application deployment in telco clouds brings an unexplored environment to edge NFV. The cloud-native environment deploys containerized applications as a loosely-coupled system (often implemented as microservices) with optimized orchestration and resource utilization to deliver extreme simplicity, scalability, and resilience. The cloud-native service inherently fits the 5G edge use cases such as vRAN and MEC considering the heterogeneous platforms at the edge and the differentiated service requirements. For example, various 5G services are delivered as differentiated network slices with specific network function host requirements. The phone slice only needs to host DU at edge servers, while hosting all other 5G core, IMS server, and WAN optimizers on the central cloud. While the mission-critical slices such as autonomous driving need all DU, user-plane 5G core, and V2X services to be hosted on edge servers. The cloud-native technology enables an effortless deployment, operation, and management of containerized 5G applications regardless of the edge location and heterogeneous hardware platform type.

978-1-7281-4545-7/19/$31.00 ©2019 IEEE

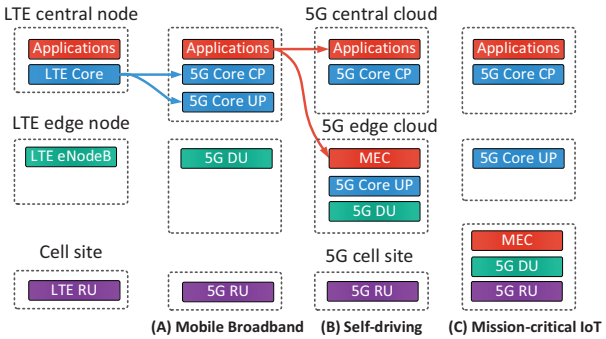


Figure 1. RAN architecture evolution from LTE to 5G.

Understanding the new edge workloads and emerging deployment manners will significantly impact the design of edge hardware platform. As the edge platform needs to seek the tradeoff between the cross-platform compatibility and the extreme energy and cost-efficiency. However, though existing work explores the virtualized BBU system under different RAN system configurations and provides initial insights on system computational capacity, more detailed architectural implications are still needed to better decide the architecture tradeoff of 5G edge cloud servers. Our goal is to provide first-hand characterization experiences.

In this paper, we build a test framework of 5G RAN and MEC based on cloud-native technology. We collect the architectural characteristics of key network components and MEC applications on the 5G edge cloud platform. Our experiments demonstrate the following implications: (1) The RAN system is a CPU consuming application, while the usage of memory is trivial. (2) The main micro-architectural bottleneck for the RAN system is caused by its Backend Bound, the optimization for Backend Bound is necessary to get better RAN performance. (3) In the RAN system, the turbo decoding module consumes the majority part of the CPU, it should be further optimized or offloaded to a hardware accelerator. (4) The co-running of the RAN system and MEC application will slow down the video processing based MEC applications by 24% ~ 100%. However, the RAN performance is not seriously affected when co-running with our chosen MEC applications.

2. 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 [1-9], there are still several NFV projects [10-13] proposed at the side of radio access networks and network edge. While the scope of the above-mentioned works includes the RAN virtualization and network slicing realization, none of them provide an architectural characterization of a virtualized RAN system, particularly as a cloud-native containerized form. Moreover, no existing work has explored the co-located vRAN and MEC systems from an architectural perspective.

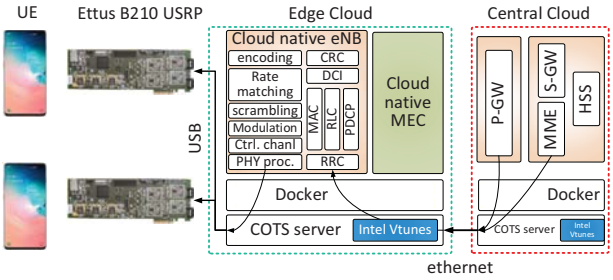


Figure 2. RAN architecture evolution from LTE to 5G.

RAN Characterization: In the last few years, several works [14-19] provide the performance analysis and study on the vRAN system. [15, 17] introduces the concepts and architecture of the vRAN system. [15] validates two MAC schedulers and analyzes the vRAN system, in terms of memory occupancy and execution time. [17] performs thorough profiling of OAI, in terms of execution time, on the user plane data flow. A recent work [23] 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 the vRAN framework. *To the best of knowledge, our work is the first one that explores the architectural implications of the colocation of cloud-native vRAN and MEC.*

3. System Architecture of 5G Edge Cloud

In this section, we first describe the evolution of RAN architecture and discuss several network function deployment cases. We then introduce our cloud-native edge cloud testbed based on open-source virtualized RAN framework OpenAirInterface (OAI) [10] and MEC Benchmark PARSEC [22].

3.1 The Architecture of 5G RAN

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 part 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 the container-enabled central cloud, edge cloud, and cell site.

We show three typical cases of network function placement corresponding to the specific application in Figure 1. For the mobile broadband service slices with roundtrip 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, an analogy to the eNodeB in LTE) is deployed on edge cloud servers. For the autonomous driving case with the roundtrip latency

guarantee is within 5ms, the 5G core data-plane functions will be deployed in edge cloud. And the edge cloud servers also host the 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.

3.2 Experimental Platform Overview

We choose the OpenAirInterface (OAI) 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 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 User Equipment (UE). For the core network, the OAI provides standard-compliant implementations of a subset of 3GPP Evolved Packer Core (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 PARSEC to mimic our video processing based MEC applications. PARSEC is a benchmark suite for studies of Chip-multiprocessors. PARSEC includes emerging applications in system applications that mimic large-scale multithreaded industrial programs. All the PARSEC applications are leveraged in containers.

3.3 Experimental Platform Setup

As shown in Figure 2, the experimental testbed consists of one/two units of Commercial-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 Intel Core machines (Core i7-8700 @ 3.20GHz 16GB RAM) for eNB DU and RU, Intel Xeon machine (E5405 @ 2.00GHz 4G RAM) for EPC and Huawei Honor 8 as our UE. The eNB version we use is branch 2018_w25. For EPC, we use the develop branch. The Operation system used for both machines is Ubuntu 16.04. The testbed is implemented with a real RF front-end (Ettus B210 USRP). All the experiments were conducted with the same eNodeB configuration, namely FDD with 5 MHz bandwidth in band 7. We use Intel VTune Amplifier [24] to profile architectural data of key network functions as illustrated in Figure 2. All the applications are implemented in the Docker container environment. The docker version we use is 18.09.1.

4. Understanding the Architecture Implications of Edge NFV Workloads

In this section, we demonstrate several key learnings that are drawn from our detailed architectural profiling. We start by characterizing the solo-run eNodeB Distributed Unit (DU). Then, we evaluate the performance of DU and edge computing applications when co-running them together on the same cores. We reason about the interesting findings and provide a platform setup guideline for cloud-native edge NFV workloads.

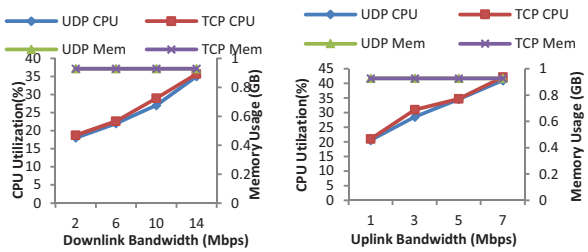


Figure 3. CPU/Memory Usage vs. UDP/TCP Bandwidth

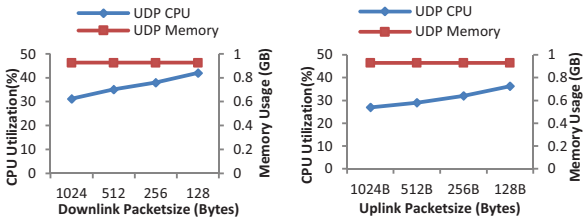


Figure 4. CPU/Memory Usage with UDP Packet size

4.1 CPU and Memory Usage of eNB

Figure 3 and Figure 4 illustrate the CPU utilization of the whole the 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) TCP – increasing bandwidth while maintaining the same packet size. (using default packet size) (3) UDP – decreasing the packet size while maintaining the same bandwidth. We choose 10Mb/s for downlink and 5 MB/s 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 pps, which illustrated that the *eNodeB Distributed Unit is a computation-intensive application*. Besides, the uplink per Mbps CPU utilization is 2 times to the downlink per Mbps CPU utilization.

4.2 CPU utilization for OAI eNodeB Sub-modules

Figure 5 and Figure 6 show the CPU utilization of sub-modules inside OpenAirInterface eNodeB for UDP traffics with different bandwidth and packet sizes. 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 layer modules.

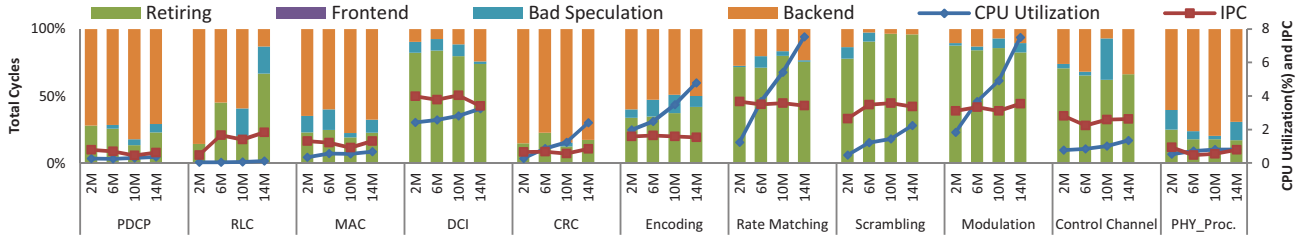


Figure 5a. CPU utilization and microarchitecture value for downlink UDP bandwidth

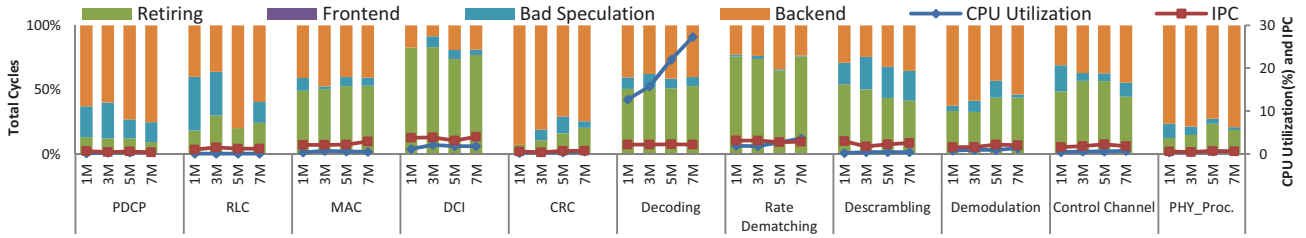


Figure 5b. CPU utilization and microarchitecture value for uplink UDP bandwidth

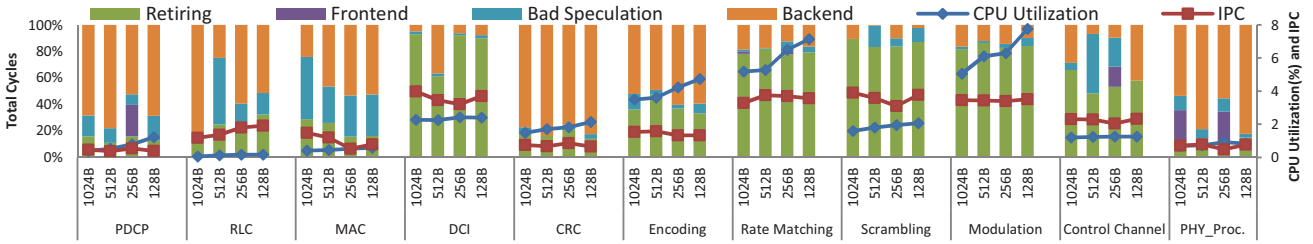


Figure 6a. CPU utilization and microarchitecture value for downlink UDP packet size

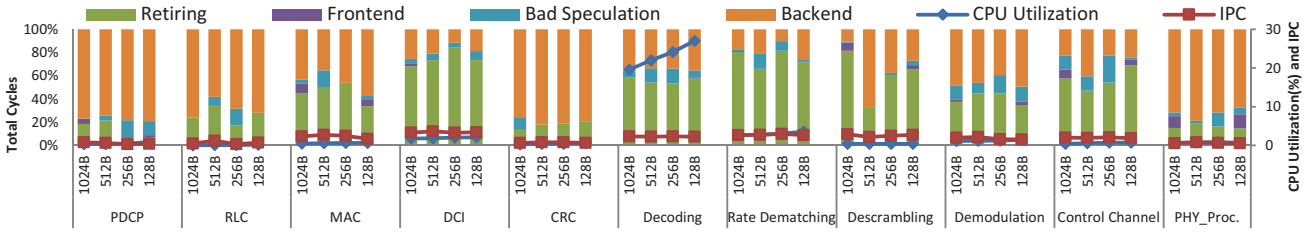


Figure 6b. CPU utilization and microarchitecture value for uplink UDP packet size

Figure 5 and Figure 6 also illustrate that the *CPU utilization of different modules inside eNodeB alters greatly in uplink and downlink direction*. For the downlink, the CPU consumptions are evenly distributed among the modules. For the Uplink, Turbo decoding consumes 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 the work [21]. *The turbo decoding module should be highly optimized or this module is suggested to implement by a hardware accelerator*.

Besides, from Figure 5 and Figure 6 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.

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 the smaller usable maximum bandwidth. The trend for CPU consumption for each module of UDP and TCP is similar to each other.

4.3 Instructions Execution

Instruction per cycle (IPC) is a fundamental performance metric indicating the average number of instructions executed for each clock cycle, which is used to measure instruction-level parallelism. There are four micro-architectural metrics relate to the IPC – Retiring, Bad Speculation, Frontend Bound and Backend Bound [20]. The high percentage of retiring usually means the high IPC value of an

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

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 to locate the hotspot of an application and provide optimization direction for further development.

Figure 5 and Figure 6 show IPC 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. Analysis of the micro-architecture bottleneck is given in section 4.4.* Figure 5 and Figure 6 also provides the Retiring percentage of modules inside OAI eNB, we note that Retiring correlates well with IPC value, high retiring value always means a high IPC.

4.4 Frontend Bound, Bad Speculation and Backend Bound Behavior Analysis

Figure 5 and Figure 6 also show the cycle breakdown for the submodules of DU. Front-end Bound denotes that instruction-fetch stall will prevent the car 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 the DU application is concentrated at Backend Bound, which means *the optimization for the 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 3. Cache Size and Frequency in Wimpy and Beefy Node

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

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 6. Edge Computing Applications

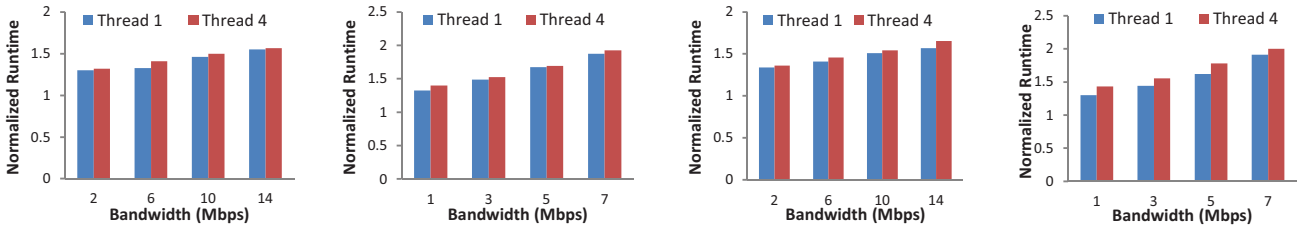
	Information	Workload
vips	Image processing	Image 18K*18K pixels
x264	Video encoding	25fps 640*360 pixels

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 2 show the Memory Bound and Core Bound details of 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 for the current RAN system. Memory Bound can be mitigated by increasing the cache size of the server. Core Bound can be mitigated with better code generation, e.g., avoiding dependent arithmetic operations in a sequence; A compiler with better instruction scheduling; Or better vectorization organization for OpenAirInterface system.*

4.5 eNodeB Performance on Beefy Server

We further deploy the DU on an alternative high-end COTS server platform with higher cache size to investigate if the backend bound overheads could be mitigated. Table 3 compares the cache size and CPU frequency of the wimpy server and the beefy server. *The beefy server we use for DU is the Intel Xeon machine (W2195 @ 2.30GHz, 128GB RAM). The compared wimpy server is original Intel Core machines (Core i7-8700 @ 3.20GHz 16GB RAM)*

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 beefy in the server environment. The counteracts of lower Memory Bound and higher Core Bound makes the overall backend performance stay similar to the wimpy server platform.



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

Figure 7. The Influence of edge computing applications' runtime when co-running with eNodeB DU

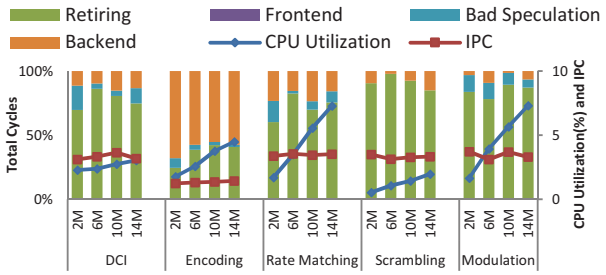


Figure 8a. CPU utilization and Microarchitecture value for Downlink UDP with vips

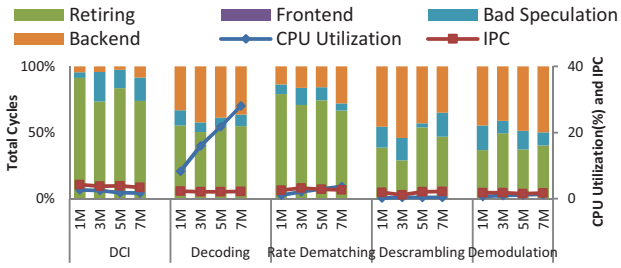


Figure 8b. CPU utilization and Microarchitecture value for Uplink UDP with vips

4.6 Co-location of RAN and MEC Application

In this section, 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 choose video processing related applications as our MEC applications since most of today's edge computing utilizations are concentrated on video processing. We choose vips and x264 from the PARSEC benchmark. 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 7 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.

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 8 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.

We speculate 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.

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 the 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 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 utiliza-

tion should be constrained when co-running with another eNodeB since they have the same level priority. However, since we only have one USRPb210 card at the time, this experiment has to remain in the future.

5. Acknowledgment

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.

6. Conclusion

We outline a comprehensive workload characterization for virtual eNodeB and edge computing in this paper. According to our workload characterization work, the main bottleneck for the virtual eNodeB is its Backend Bound; the optimization should be the focus on its memory utilization and core port utilization. Besides, when the eNodeB is co-running with the edge computing applications, the edge computing applications are slowed down by the eNodeB application. However, the performance of the eNodeB is not affected by our chosen edge computing applications.

References

[1] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, Yongguang Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," Proceeding Co-NEXT '10，Philadelphia. Pennsylvania, Article No. 15.

[2] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, Ronghua Zhang, "Network Virtualization in Multi-tenant Datacenters," Proceeding NSDI' 14, Seattle. WA, April 2014, pp. 203-216.

[3] Dmitry Drutskoy, Eric Keller and Jennifer Rexford, "Scalable Network Virtualization in Software-Defined Networks," Journal IEEE Internet Computing, March 2013, pp. 20–27.

[4] Malla Reddy Sama, Xueli An, Qing Wei and Sergio Beker, "Reshaping the Mobile Core Network via Function Decomposition and Network Slicing for the 5G Era," 2016 IEEE Wireless Communications and Networking Conference, Doha. Qatar, April 2016.

[5] Ravishankar Ravindran, Asit Chakraborti, Syed Obaid Amin, Aytac Azgin, and Guoqiang Wang, "5G-ICN: Delivering ICN Services over 5G Using Network Slicing," IEEE Communications Magazine

[6] Miloud Bagaa, Tarik Taleb, Abdelquoddouss Laghrissi, Adlen Ksentini, and Hannu Flinck, "Coalitional Game for the Creation of Efficient Virtual Core Network Slices in 5G Mobile Systems," IEEE Journal on Selected Areas in Communications, March 2018,

[7] Hu, Yang, Mingcong Song, and Tao Li. "Towards full containerization in containerized network function virtualization." ACM SIGOPS Operating Systems Review 51, no. 2 (2017): 467-481.

[8] Hu, Yang, and Tao Li. "Towards efficient server architecture for virtualized network function deployment: Implications and implementations." In The 49th Annual IEEE/ACM International Symposium on Microarchitecture, p. 8. IEEE Press, 2016.

[9] Hu, Yang, and Tao Li. "Enabling efficient network service function chain deployment on heterogeneous server platform." In 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 27-39. IEEE, 2018.

[10] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, Christian Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," ACM SIGCOMM Computer Communication Review, October 2014

[11] Qinghua Zheng, Haipeng Du, Junke Li, Weizhan Zhang, "Open-LTE: An Open LTE Simulator For Mobile Video Streamin," 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), Chengdu. China, July 2014.

[12] Antonio Virdis, Giovanni Stea, and Giovanni Nardini, "SimuLTE — A Modular System-level Simulator for LTE/LTE-A Networks based on OMNeT++," Proceeding SIMULTECH, Vienna. Austria, August 2014, pp. 59-70.

[13] Zhang, Lu, Chao Li, Pengyu Wang, Yunxin Liu, Yang Hu, Quan Chen, and Minyi Guo. "Characterizing and orchestrating NFV-ready servers for efficient edge data processing." In Proceedings of the International Symposium on Quality of Service, p. 22. ACM, 2019.

[14] Antonio Virdis, Niccolo Iardella, Giovanni Stea and Dario Sabella, "Performance analysis of OpenAirInterface system emulation," 2015 3rd International Conference on Future Internet of Things and Cloud, Rome. Italy, October 2015.

[15] Chun Yeow Yeoh, Mohammad Harris Mokhtar, Abdul Aziz Abdul Rahman and Ahmad Kamsani Samingan, "Performance study of LTE experimental testbed using OpenAirInterface" 2016 18th International Conference on Advanced Communication Technology (ICACT), Pyeongchang. South Korea, March 2016.

[16] Oumayma Neji, Nada Chendeb, Olfa Chabbouh, Nazim Agoulmine and Sonia Ben Rejeb, "Experience deploying a 5G C-RAN virtualized experimental setup using OpenAirInterface," 2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband (ICUWB), Salamanca. Spain, January 2018.

[17] Po-Chiang Lin and Sheng-Lun Huang, "Performance Profiling of Cloud Radio Access Networks using OpenAirInterface," 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), HI. USA, March 2019.

[18] Qi Zheng, Yajing Chen, Ronald Dreslinski, Chaitali Chakrabarti, Achilleas Anastasopoulos, Scott Mahlke, Trevor Mudge, "WiBench: An open source kernel suite for benchmarking wireless systems", 2013 IEEE International Symposium on Workload Characterization (IISWC), Portland, OR, USA, Sept. 2013

[19] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton , Pablo Serrano, Cristina Cano, Doug J. Leith, "srsLTE: an open-source platform for LTE evolution and experimentation", Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization WiNTECH, New York City, New York, October , 2016

[20] Ahmad Yasin , "A Top-Down method for performance analysis and counters architecture", 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, March 2014

[21] Vanchinathan Venkataramani, Aditi Kulkarni, Tulika Mitra, Li-Shiuan Peh "SPECTRUM: a software defined predictable many-core architecture for LTE baseband processing", 2019 Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), Phoenix, AZ, USA, June, 2019

[22] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, Kai Li, "The PARSEC benchmark suite: characterization and architectural implications", Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT), Toronto, Ontario, Canada, October, 2008

[23] Tuyen X. Tran, Ayman Younis, Dario Pompili, "Understanding the Computational Requirements of Virtualized Baseband Units Using a Programmable Cloud Radio Access Network Testbed", 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, USA, July， 2017

[24] Intel Vtune Amplifier: https://software.intel.com/en-us/vtune