Enhancing The Design Of NextG For Critical And Massive IoT Devices And Applications

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Udhaya Kumar Dayalan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Zhi-Li Zhang

February, 2024

Acknowledgements

There are lot of things to be thankful for and appreciate in this life and in my PhD journey. In this enriching journey, I am profoundly grateful for the unwavering support and inspiration from colleagues, teachers, friends and family.

Advisor. First and foremost, my deepest thanks to **Prof. Zhi-li Zhang** for guiding me through the intricate path of academia and for his unwavering guidance in shaping my academic journey. Even though I was a part-time student with a full-time job and 2 kids, Professor gave me the same support in everything, including his time and supporting me to attend technical courses and travels to conferences. His passion towards research and aim to conduct quality research was a great inspiration to me. During these years, I was lucky enough to see the other beautiful side of Professor, his love and compassion for his students. I always felt that he is always there for me, such a humble and kind human being. I extend my heartfelt gratitude for his invaluable insights, constructive feedback and mentorship throughout my PhD journey.

Manager/Mentor. My sincere thanks to my manager, **Mr. Shane Gydesen**, whose unwavering support extended beyond the professional realm, making a significant impact on my academic pursuits. He is a biggest factor in my academic and career success. He never see his reportee's as just a colleague, he invests on their career and growth. He always makes sure that I do a proper wok-study-life balance. He was always there for me during both good and bad times. A heartfelt thank you for his exceptional support that transcended the boundaries of the workplace.

Committee. My sincere gratitude to the esteemed members of my PhD Thesis Committee: **Prof. Zhi-Li Zhang, Prof. Nikolaos Papanikolopoulos, Prof. Kumar Mallikarjun, Prof. Ali Anwar, and Prof. Feng Qian**. Your time, feedback, and motivation were instrumental in shaping this research.

Grants. I would like to acknowledge the grants that supported my research. This research was supported in part by the National Science Foundation (NSF) under Grants CNS-1814322, CNS-1836772, CNS-1831140, CNS-1901103, CNS-2106771 and CNS-2128489, and Seed Grants from University of Minnesota MnRI, CTS and CSE and an unrestricted gift from InterDigital.

UMN Networking Lab. I would also like to acknowledge the exceptional camaraderie and collaboration with my friends at UMN Networking Lab: **Dr. Eman Ramadan, Mr. Gaurav Gautham, Ms. Ngan Nguyen, Ms. Xinyue Hu, Mr. Rostand A. K. Fezeu, Mr. Jason Carpenter, Mr. Wei Yu, Mr. Feng Tian, Mr. Nitin Varyani, Mr. Ziyan Wu, Dr. Arvind Narayanan, and Mr. Timothy J. Salo**. Your support, collaboration, and shared time have been invaluable.

University of Minnesota. Thanks to the University of Minnesota for the teachings, learning, and cherished memories since 2016. Special acknowledgment to my Masters (MSSE) advisor, **Dr. Mike Whalen**, and **Prof. Mats Heimdahl** for their support. I like to thank **Joseph (Joe) Nieszner**, Graduate Programs Manager, for his continued support, he was always there for me and other students.

Trane Technologies. I extend my gratitude to my current employer, Trane Technologies, for playing a pivotal role in my PhD journey. Thanks to **Mr. Brady Moroney** and **Mr. Anil Gopinathan** for their unwavering support and motivation. Thanks to my friends **Mr. Jerome Beski**, **Mrs. Ann Arora**, **Mr. Nate Longen**, **Mr. Jim Mckeever** and **Mr. Tom Basterash** for all their love and support.

Minnesota. Minnesota holds a special place in my heart, and I am thankful for all the beautiful memories since 2008.

Friends. To my Minnesota friends, your support during challenging times, especially when my family was away due to my son's health situation, will forever be cherished. The food and love you shared are eternally appreciated. The time I spent with you all helped me to recover and recoup during challenging times. Looking forward to make more memories with you all in the upcoming years. I owe a lifelong debt of gratitude to my school and college friends, particularly **Mr. Sathish Vasu**, a true gift, a selfless person who was always there for me when I had nothing, who generously provided his computer for my learning and college projects. My friend **Mr. Selvaraj**, who inspired me and has been a constant support throughout my college life and beyond.

Family. I express my deepest appreciation to all family members, especially my brother **Mr. Kirubakaran & family**. You were a biggest support for me during challenging times while pursuing my undergraduate. Special thanks to my father(-in-law) **Mr. Rajasekaran & family**. You are my role-model and a wonderful mentor. Thanks to my co-brother **Mr. Raman & family**, for their unwavering support.

Parents. To my dad, **Mr. Dayalan**, your wisdom resonates, and I hold no regrets—only love and joy for everything you have given me in my life. Even though, you were not able to go to college, you made sure that both your kids got good education from reputed institutions. Mom, **Mrs. Santhanalakshmi**, your sacrifice and unconditional love are beyond words. If you were not there, I would have not been an engineer now. You sow the seed of importance of education deeply into me. You taught me everything in my life and you are my first best friend. Instead of investing on materials things, I am proud that my parents invested on the education of their kids.

Kids. Thanks and love to my kids. **Ms. Varshini**, your understanding, support, and maturity have been a pillar for our family. You took good care and control of your studies, so that I can focus on my PhD. **Master. Pranav**, you deserve all my time post-PhD. Both of your faces were the source of inspiration and your touch were the source of healing for me when I had challenging times during my PhD journey.

My Source. Well, I cannot say, first of all or finally for this special person in my life because she exists throughout anything and everything of my life, my wife, **Mrs. Lakshmi**. You've been a constant source of selfless, unconditional love. Words cannot capture the depth of gratitude I feel. You were with me, held me and supported me when I had nothing in my life. You brought the good stuff out of me and guided me to uncover my true potential. You made sure I ate good food, practice yoga regularly, meditate frequently and stayed healthy, so that I can focus on my work and PhD. You took the ownership, led our family from front and took additional responsibilities, so that I can focus on my studies. Thanks a lot for everything you have done for me and our family all these years. Look, what you have made me, Dr. Udhaya Kumar Dayalan!

Dedication

To my awesome wife Mrs. Lakshmi who is a big support for me since when I was/had nothing and guided me in each aspects of life. To my beautiful kids Ms. Varshini and Master. Pranav to whom I owe time. To my lovely parents Mr. Dayalan and Mrs. Santhanalakshmi who gave me the wings to fly, freedom to think and try new things. To my family and friends who held me up over the years.

Abstract

The IoT world is evolving with the latest technology trends like edge computing, augmented & virtual reality, machine learning, robotics and 5G. But still there is less business productivity due to the slower technical adoption in the industrial automation system. There is a tremendous need for autonomous networks in the manufacturing industry to increase productivity and allow communication between people, devices and sensors. And there are massive numbers (hundreds to thousands) of IoT devices in a single factory depending on the scale of the industry. These factories consist of critical IoT devices like fire or gas sensors which need to operate reliably with less latency. But the existing wired and/or wireless networks are struggling to fulfill the computational and resources for operations demand of the emerging technologies. In order to address these needs of the industries with digital transformation happening in Industry 4.0, the evolution of private 5G and 5G standards opens bigger opportunities.

In this thesis, we discuss the challenges and explore new opportunities of using 5G for critical/massive IoT devices. While exploring the possibilities, we uncover some of the challenges in IoT especially due to the vendor-locked IoT solutions and unavailability of large scale IoT devices for evaluating end-to-end systems. We discussed the challenges in detail and proposed novel solutions to overcome these challenges.

First, as the plethora of Internet of Things (IoT) devices gradually make their way into our lives, several Cloud Service Providers (CSPs) have developed *IoT gateway* platforms (SDKs) that solely connect IoT devices to their respective cloud. Such gateways are *cloud-centric*. We study the state-of-the-art vendor-locked IoT Gateway solutions and approaches and propose an *edge-centric* paradigm through an evolutionary framework, dubbed *VeerEdge* for developing *IoT gateways*. We leverage computing and storage capabilities at the network *edge* for edge-based device & IoT service.

Second, the IoT world is evolving with the latest technology trends like edge computing, augmented & virtual reality, machine learning, robotics and 5G. With the digital transformation happening in Industry 4.0, many industries are moving towards private 5G networks. There are a massive number (hundreds to thousands) of IoT devices in a

single factory depending on the scale of the industry and these factories consist of critical IoT devices like fire or gas sensors which need to operate reliably with less latency. In order to efficiently realize the capabilities such as ultra reliable low latency communications (URLLC), enhanced mobile broadband (eMBB) and massive machine-type communications (mMTC) offered by 5G, the next generation IoT devices/applications need a paradigm shift in their design and need to be evaluated under simulation using 5G networks before getting deployed in the real-world. However, many IoT simulators run in isolation and do not interface with real-world IoT cloud systems or support 5G networks. This isolation makes it difficult to design, develop and evaluate IoT applications/devices for industrial automation systems and for experiments to fully replicate the diversity that exists in end-to-end, real-world systems using 5G networks. Kaala 2.0 is the first scalable, hybrid, end-to-end IoT and NextG system simulator that can integrate with real-world IoT cloud services through simulated or real-world 5G networks. Kaala 2.0 is intended to bridge the gap between IoT simulation experiments and the real world using 5G networks. The simulator can interact with cloud IoT services, such as those offered by Amazon, Microsoft and Google. Depending on the configuration, Kaala 2.0 supports simulation of User Equipment (UE), 5G Radio Access Network (RAN) and 5G Core and at the same time supports real-world User Equipment (UE), 5G Radio Access Network (RAN) and 5G Core. Kaala 2.0 has the ability to simulate a large number of diverse IoT devices to evaluate mMTC, simulate events that may simultaneously affect several sensors to evaluate URLLC and finally simulate large amounts of data to evaluate eMBB.

Third, we argue that existing 5G network architecture is too rigid to support many future applications with high bandwidth and low latency. This is in spite of the O-RAN vision to endow radio access networks (RAN) with agility and intelligence via RAN intelligent controllers (RICs). We posit that not all data is of equal utility to applications, and advocate an (application) *semantics-aware*, *fine-grained*, *cross-layer* and *software-defined* framework to re-architect next-generation (NextG) networks. We focus on the design of HyperRAN, an intelligent NextG RAN architecture that embeds application semantics across the RAN protocol stack to enable agile and intelligent decision making. At the core of HyperRAN is a *declarative, programmable* Hyper Scheduler that takes into account application semantics, service requirements, user context as well as channel

conditions for intelligent and adaptive radio resource scheduling.

Finally, we advocate an eBPF (extended Berkeley Packet Filter)+XDP (eXpress Data Path) based framework for scaling and accelerating software packet processing in (O-RAN compliant) NextG RANs. Using 5G Central Unit User Plane (CU-UP) as a key case study, we present an initial design of our proposed framework, dubbed PRANAVAM, and its key components. We also discuss additional design and options for further improvements.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unlike earlier generations of cellular technologies, emerging 5G networks are designed to enable a whole gamut of diverse new use cases from massive consumer/industrial IoT devices to control, safety and other V2X (vehicle-to-everything) applications for autonomous vehicles (AVs) and drones to ultra-high-resolution (8K & volumetric) live video streaming, augmented/virtual reality (AR/VR), telemedicine and healthcare. These new use cases are categorized as mMTC (massive machine type communication), ULLRC (ultra-low latency reliable communication) services and eMBB (enhanced mobile broadband). Emergent 5G networks introduce a wide spectrum of radio bands, from 5G *low-band* (sub-1GHz spectrum bands) to 5G *mid-band* (1 GHz – 7.125 GHz frequencies) and 5G *high-band* (24GHz – 60 GHz) including mmWave radio bands. These spectrum radio bands coupled with the so-called flexible numerology[2] and frame structure, dynamic single/mini-slot scheduling (with dedicated Downlink (DL) and Uplink (UL) symbols) and semi-persistent scheduling (SPS), slot configuration and aggregation with preemptive scheduling mechanisms to support ultra-low, ultra-reliable and ultra-high bandwidth applications. Likewise, much of the *architectural designs* of 5G radio network (RAN) and core network (5GC) are driven by the needs to support these new use cases by utilizing a wide and diverse range of spectrum bands.

These 5G promises and new capabilities trigger an important question: Is 5G's unique and complex architectural design and proposed flexible features "*Enough*" to support envisioned 5G applications and services, some of which have high reliability,

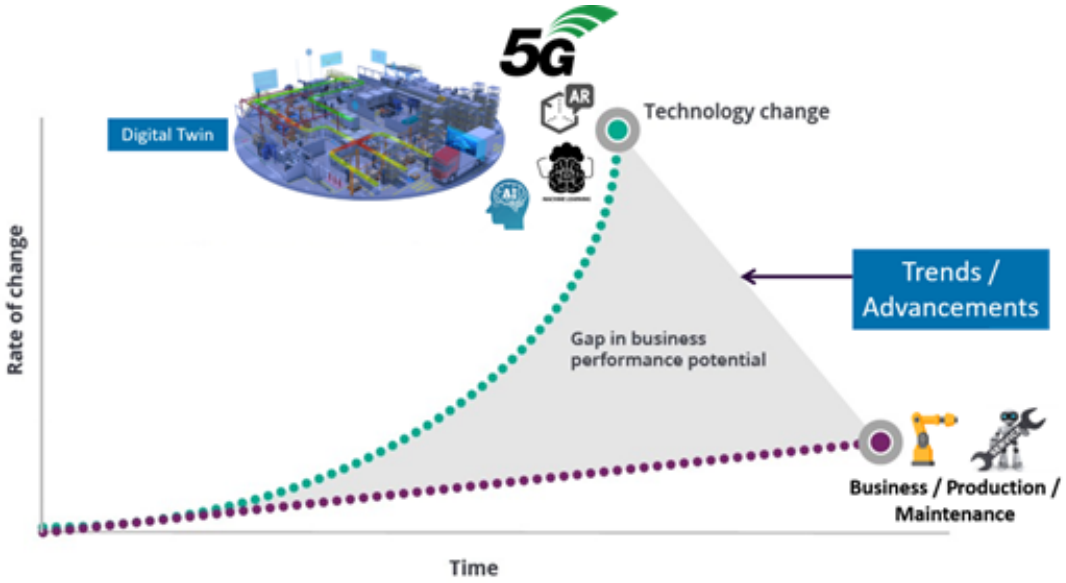


Figure 1.1: Gaps in Manufacturing Industry & Trends [1]

confidentiality and security demands, others demand ultra-high bandwidth and ultra-low latency with massive connectivity and supreme user quality of experience, or any combination of the above? Specifically, can 5G's high sensitivity to obstruction, especially mmWave 5G (e.g., moving cars/people, building, trees etc.) support existing video streaming services? Can 5G help manufacturing industry to catch-up with the technology changes shown in Fig. 1.1?

Intelligence and agility are part of the goals driving the O-RAN vision postulated by the O-RAN Alliance [3]. Apart from a dis-aggregated architecture and open APIs, O-RAN aims to make radio access networks (RAN) more agile and "intelligent" by introducing (non-real-time and near-real time) RAN intelligent controllers (RICs) (§2.3). While "softwarization" or "cloudification" enabled by RAN disaggregation makes it easier to introduce new features, e.g., incorporation of artificial intelligence (AI) models, the extent to which RICs can enable 5G RAN to make intelligent decisions is still fundamentally limited by the existing 5G RAN capabilities (see §5.2 for discussion). For example, when the aggregated bandwidth demand exceeds the available radio resources, no AI-guided RICs can magically resolve this fundamental *demand-resource mismatch* problem.

The plethora of *Things* surge in recent years has more than ever influenced the IoT Ecosystem: An end-to-end connected system of *Things*, operating systems, gateways, middleware and cloud platforms [4]. This influx of *Things* drives researchers to constantly look for new solutions to support service deployment velocity and integration [5, 6, 7, 8, 9]. As these *Things* become an integral part of our lives, *dependability* of IoT services within an IoT Ecosystem is paramount. Despite the prominence of an interoperable IoT gateway on the *dependability* of IoT services, current gateways still suffer from major challenges which restricts massive deployment and limited installation in practice. To emphasize these challenges: 1) Current gateway solutions are equipped with multi-protocol parsers, each which support specific IoT protocols. This hinders massive deployment considering the diversity of custom protocols within an IoT Ecosystem. 2) Gateway solutions are vendor-specific. *i.e.*, With different architectures, unique IoT service functions and proprietary APIs [10]. The management of several diverse unique end-to-end IoT deployment solutions is not ideal within the IoT Ecosystem. 3) Current gateways have isolated scopes of supported devices. *i.e.*, Each vendor gateway deployment advocates a list of supported devices, making deployment locked and limited, particularly in a diverse end-to-end cross-vendor deployment scenario.

The number of IoT devices in a building scales from 1 to more than thousands depending various factors like the size of the building, number of people working or accessing that building every day, and the type of operations performed in the building (like data centers, call centers, universities, hospitals, IT companies, etc.). It is challenging to design and test different types of IoT devices [11] or to analyze and benchmark performance and scalability of a large IoT Ecosystem [12]. Although we have few simulation frameworks as listed in [13], none of the simulation frameworks provide simulation of multi-vendor specific IoT devices [13]. And also the vendor-specific simulation option given by the respective vendor CSP [14], can simulate only a single device and that is specific to that vendor only [14]. And none of the simulation frameworks provide simulation of multi-vendor specific IoT devices [13].

1.1 Outline and Contributions

In this thesis, we propose enhancing the design of NextG for critical & massive IoT devices and applications. The outline of this dissertation, along with the primary contributions of this dissertation are as follows:

In Chapter 2, we provide a review for required background knowledge including IoT system architecture, 5G networks and O-RAN architecture. Moreover, we introduce the detailed background which motivates our works of this dissertation.

In Chapter 3, we study the state-of-the-art vendor-locked IoT Gateway solutions and propose an *edge-centric* paradigm through an evolutionary framework, dubbed *VeerEdge* for developing *IoT gateways* that exploits the availability of multiple cloud services, storage and computing capabilities on the network for edge-based device management and configurations and "best" IoT data analytics. We investigate a critical IoT gateway functionality dubbed - *Regulator* that realizes the edge-centric gateway vision by controlling the communication between vendor-specific IoT gateways and their respective cloud services.

In Chapter 4, we introduce *Kaala 2.0*, a scalable, hybrid, end-to-end IoT system simulator that is able to create IoT devices of various types to communicate with real-world cloud IoT systems, with AWS, Amazon and Google IoT Cloud platforms as case studies. Kaala is a scenario-based IoT simulator capable of mimicking various IoT scenarios such as "fire in a room or building" and "5G network capable data generation (including 4K/8K video IP cameras)" scenarios. Kaala 2.0 is able to generate massive amounts of IoT data for prototyping data-intensive IoT applications.

In Chapter 5, we lay out an overall framework for re-architecting NextG networks. Our framework enables application endpoint and network collaboration by (i) (adaptively) refactoring, partitioning and marking application data with *semantic tags* and embedding them end-to-end across the network and down the network protocol stack; and (ii) endowing NextG RAN with the agility to intelligently match available frequency channels with differing characteristics to appropriate data (sub-)streams/objects, and dynamically allocate fast varying radio resources to transport the right (amount/type of) data with the best deliverable utility to an application.

In Chapter 6, we present an eBPF+XDP-based framework, dubbed PRANAVAM,

for (O-RAN compliant) future RAN architecture development. Using 5G CU-UP as a key case study, we outline the initial design of our proposed PRANAVAM. Using eBPF+XDP for kernel extension/bypassing, our preliminary evaluation shows that PRANAVAM improves the throughput by 22-26% over existing 5G RAN implementations. We also discuss an additional design, PRAVEGA, in which the GTP-U packets are completely handled in the kernel space without being sent to the user space. We also discuss additional options to further accelerate software packet processing to scale 5G RAN implementation to meet bandwidth and latency demands.

In Chapter 7, we present concluding remarks, lessons learned, and thoughts for the future.

Chapter 2

Background & Motivation

We provide a brief overview of 5G networks, 5G RAN protocol stack and O-RAN architecture. We refer the reader to 3GPP specifications [15, 16, 17] and O-RAN specifications [3] for more details. Next, we briefly describe IoT system architecture, IoT gateway architecture and interoperability. We end by a brief discussion of eBPF/XDP.

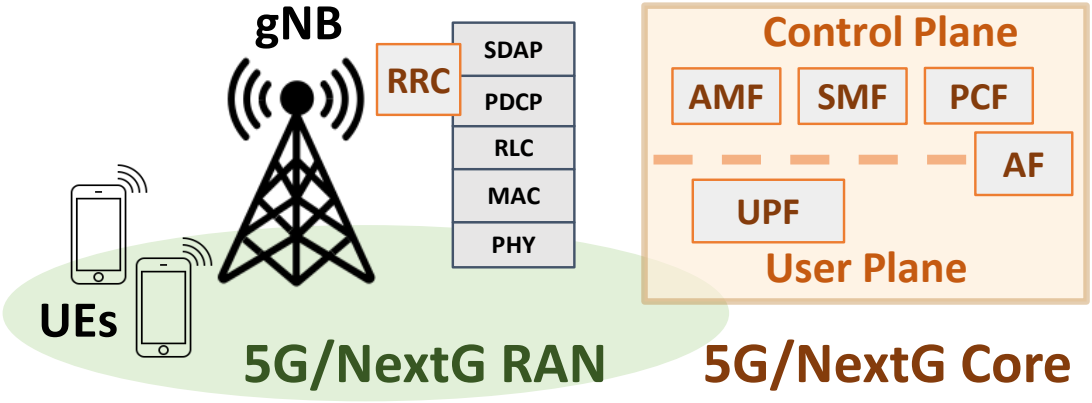


Figure 2.1: 5G RAN and Core Networks.

2.1 5G Networks: 5G NR, RAN and Core

As illustrated in Fig. 2.1, a 5G network consists of 5G RAN and 5G core. 5G RAN retains largely the same protocol architecture as 4G LTE RAN – the main difference lies in the introduction of a new SDAP sublayer at the top of its RAN protocol stack

to support the so-called flow-based QoS framework, which we will further discuss in §2.2. The bottom 4 sublayers – PHY, MAC, RLC and PDCP – performs the same functionality as those in 4G LTE. As in 4G LTE, RRC (radio resource control), part of the RAN control plane (CP), is responsible for the configuration and control of the RAN protocols. Besides the addition of SDAP, the main innovations in 5G RAN standards lie primarily in 5G NR. 5G NR introduces wider channel bandwidths, flexible numerology (subcarrier spacing), enhanced beamforming, MIMO (Multiple Input, Multiple Output), and CA capabilities, among other improvements. These new features offer the potential for significantly higher throughput (e.g., up to 1s and perhaps 10s of Gbps), potential for ultra-low latency (e.g., sub-milliseconds) and ultra-reliable services. These in turn lead to the promise (or "hype") of many new use cases.

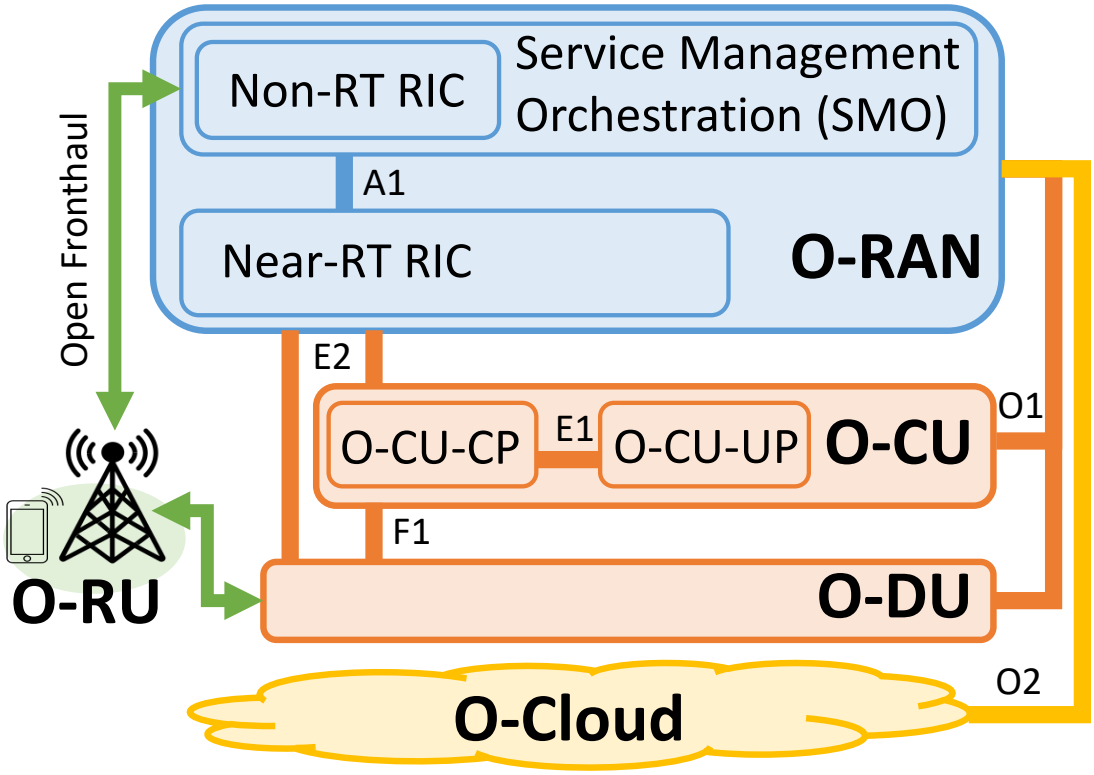


Figure 2.2: O-RAN Arch. and APIs.

In contrast to 4G LTE, 5G core is re-architected by fully embracing "virtualization" (or "softwarization" or "cloudification"): both CP and user plane (UP) are structured

into a set of (network) functions. In particular, 5G control plane comprises various control functions such as AMF (Access Management Function), AUSF (Authentication Server Function), SMF (Session Management Function), and PCF (Policy Control Function), which are responsible for managing access, authenticating users, establishing and configuring protocol data unit (PDU) sessions. User data packets or PDUs (protocol data units) between UE and a PDN (e.g., the public Internet) are carried in PDU sessions and processed by one or more UPFs (user plane functions) in the 5G data plane.

2.2 Channels, Data Radio Bearers and 5G Flow-based QoS Framework

We now delve into some specifics of the 5G *flow-based* QoS framework, which is built on top of the existing 4G LTE *class-based* or *bearer-based* framework. Hence understanding how the latter works is crucial. We note first that 3GPP specifications introduce several notions of "channels" that are used to carry data (both UP traffic as well as RRC and cellular core *Non-Access Stratum* (NAS) – i.e., control plane – messages across PHY, MAC and RLC sublayers. At the very bottom are *physical (radio) channels* that actually transmit/carry data (after the PHY sublayer processing) in the form of radio waves. *Transport channels* and *logical channels* can be viewed as interfaces between PHY and MAC, and MAC and RLC sub-layers, respectively: PDUs (protocol data units) of each logical channel from the RLC sublayer are multiplexed into MAC frames, which are segmented into *variable-size* transport blocks and then passed down to the PHY sublayer. The size of each transport block hinges primarily on the physical radio channel characteristics and thus the modulation and coding scheme (MCS) used, but also on the availability of data queued at the logical channel. *RLC channels* serve as the interface between RLC and PDCP. In 5G, each RLC channel is configured with one of the three modes, *Transparent* Mode (TM) for RRC/NAS control messages, *Unacknowledged* Mode (UM) that can be used for user traffic that do not require reliability, e.g., voice traffic, and *Acknowledged* Mode (AM) that are used by default for most user data.

Data radio bearers (DRBs) are entities in the PDCP sublayer to "carry" user data[1],

[1]The corresponding entities for "carrying" RRC (and encapsulated NAS messages from the core)

and they are where QoS treatments are applied. When establishing a PDU session for a user (or rather, UE), a DRB is created in PDCP, and a certain QoS class as specified by the QCI (QoS class identifier) may be associated with it. 3GPP has pre-defined a set of QCIs with *fixed* QoS parameters such as guaranteed or non-guaranteed bit rates (GBR/non-GBR) priority level, packet error/loss rate, data burst volume, etc. For example, certain QCIs are defined for conversational voice, real-time gaming, live video streaming. QCI value 9 is typically used for the *default* DRB (thus with no QoS provided).

5G builds on 4G QCIs, now referred to as 5GIs (5G QoS identifiers) [18]. The main difference lies in the introduction of the SDAP sublayer which enables a "flow-based" QoS framework [19]. The new SDAP header contains a 6-bit QoS Flow Identifier (QFI) field as well as 1-bit reflective QoS Indication (RQI) and 1-bit RDI (reflective QoS flow to DRB mapping indication). Now downlink TCP/UDP 5-tuple flows may be classified at 5G core UPFs into *QoS flows* based on a set of packet detection rules (PDRs) supplied by 5G PCF. Once entering the 5G RAN, SDAP assigns each user QoS flow with a corresponding QFI based on configured rules and maps them to (pre-established) DRBs with appropriate 5QI parameters. (If RQI is set, SDAP on the UE side will use the same QFI value for uplink (UL) flow as is in the (downlink (DL) flow.) Hence 5G still relies on DRBs for QoS treatments[2] as in 4G. In other words, from PDCP and below, 5G QoS is not different from 4G QoS – DRBs with (pre-defined QoS parameters) fundamentally determine how user data will be treated at the lower RAN sublayers, e.g., scheduling weights used by the MAC scheduler for radio resource allocation. What the 5G flow-based QoS framework provides is the ability to map different user (QoS) flows to different DRBs at the 5G RAN. We will further expand on the limitations of the 5G flow-based QoS framework in §5.2. The relations among various notions of channels, DRBs and QoS flows are depicted in Fig. 2.3.

---

control messages are signal radio bearers (SRBs), which are mapped to RLC channels configured with TM.

[2]Such QoS treatments are typically implemented in the PDCP sublayer using leaky-bucket rate control, traffic shaping, admission control and other mechanisms. These mechanisms as well as how DRBs are configured (e.g., what QCI parameters are supported or can be configured) are all *vendor-specific*, *closed* and difficult, if not infeasible, to be dynamically programmed.

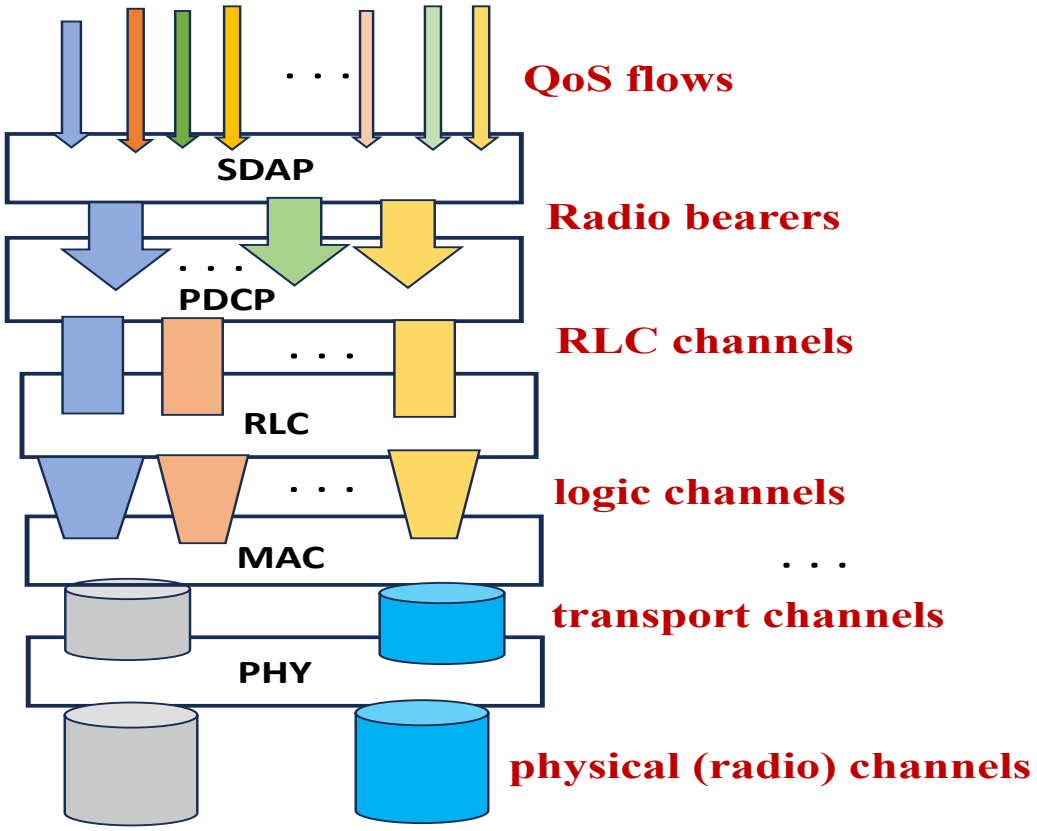


Figure 2.3: Relations among Channels, DRBs, and QoS Flows.

2.3 RAN Dis-Aggregation and Open-RAN

Starting from 4G, RAN *dis-aggregation* has been studied by 3GPP, and is now standardized by the O-RAN Alliance. As shown in Fig. 2.2), the monolithic 5G gNB is disaggregated into three components: *centralized unit* (O-CU) which runs the upper (sub)layer functions of the 5G RAN protocol stack such as SDAP, PDCP and RRC, *distributed unit* (O-DU) that runs RLC, MAC and upper PHY layer functions, and *radio unit* (O-RU) that consist of radio frequency (RF) front-end and antennas and runs the lower PHY layer functions. O-CU is further split into O-CU-CP (control plane) and O-CU-UP (user plane). With such a disaggregated RAN architecture, O-CU and O-DU are envisaged to be implemented as software modules running on commodity servers that form part of the (edge or back-end) cloud infrastructure (referred to as

O-Cloud. Further, O-RAN has introduced an orchestration and control framework for managing, configuring and controlling the O-CU, O-DU and O-RU with open, standardized APIs. The O-RAN architecture (see Fig. 2.2) includes i) a *Service Management and Orchestration* (SMO) framework containing a *Non-Real-Time RAN Intelligent Controller* (Non-RT RIC) implemented as rApps; and ii) a *Near-Real-Time RAN Intelligent Controller* (nrt-RIC) implemented as xApps. The O-RAN alliance has defined various APIs such as A1, E2, O1 between SMO, nrt-RIC, O-CU, O-DU, and other components (Fig. 2.2).

As pointed out in Chapter 1, "softwarization" or "cloudification" enabled by the O-RAN disaggregated RAN architecture makes it easier to introduce new features in the RAN designs, and provides more flexibility in deploying customized RANs to meet QoS requirements of specific use cases. However, while O-RAN RICs supposedly introduce "intelligent control" of RAN, e.g., via (re-)configurations of RAN parameters or features at the non- or near-real time basis, the extent of such "intelligent control" is fundamentally limited by the existing 5G RAN capabilities[3].

2.4 5G RAN Protocol Stack

Fig. 2.4 depicts the 5G RAN protocol stack specified by 3GPP, which resides below the OSI network layer ("IP layer"). 5G RAN functions are traditionally performed by *dedicated* (and *closed*) physical appliances (5G "base stations"), i.e., 5G nodeB (gNB) (see Fig.2.5), that are supplied by cellular equipment vendors. 3GPP also introduces a CU-DU split RAN architecture which is adopted by O-RAN: under this split, CU performs the upper layer functions of the RAN protocol stack, namely, SDAP, PDCP and RRC (radio resource control) layers; whereas DU performs the lower layer functions, namely, RLC (reliable link control), MAC (media access control) and PHY (physical) layers. While 3GPP also discussed multiple options for possibly further splitting the lower layer functions of DU, e.g., between MAC and PHY or upper and lower parts of the PHY layer, they were not pursued further by 3GPP. Instead, O-RAN adopts a

[3]As a case in point, both Phase I and Phase II use cases for O-RAN are mostly limited to RAN management and traffic steering related issues [20], instead of enabling new use cases. As an industrial forum, the O-RAN Alliance currently concerns itself mostly with *inter-operability* challenges via *interface standardization* instead of *architectural innovations*.

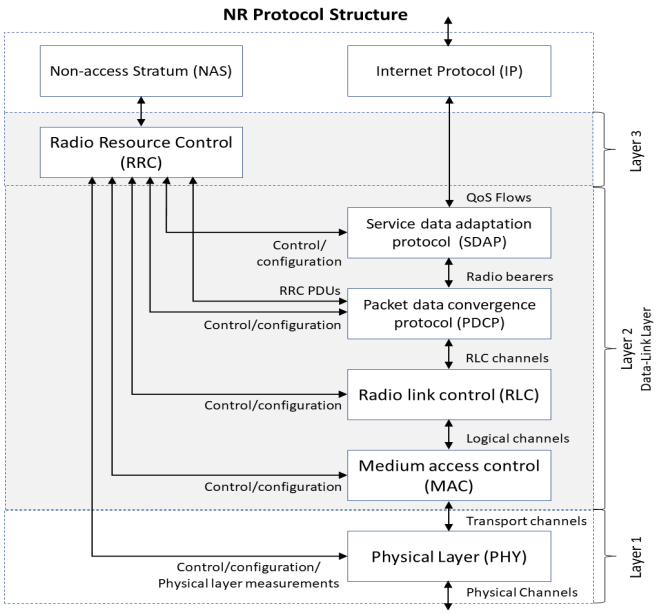


Figure 2.4: 5G Radio Access Network Protocol Stack.

version of the latter option and standardizes it. Under the so-called *7.2x split* specified by O-RAN, DU performs the RLC, MAC and the upper part of the PHY layer functions, whereas RU (radio unit) performs the lower part of the PHY layer functions. O-RAN further split CU along the control and user (data) plane separation, and introduces two components: CU-CP which performs RRC and PDCP control plane functions, and CU-UP which performs SDAP and PDCP user plane functions. We refer the reader to 3GPP specifications [15, 16, 17] and O-RAN specifications [3] for details. In this thesis, we assume a *disaggregated* RAN architecture that follows the O-RAN standard, and thus the disaggregated units are O-CU (O-CU-CP/O-CU-UP), O-DU and O-RU (see Fig.2.6, where we indicate *select* standardized interfaces *between the key units of interest*[4]). Subsequently, we will drop the prefix "O-" for clarity.

As depicted in Fig.2.6, CU-UP typically connects to multiple UPFs (via the NG interface) on the 5G core side, and may connect to multiple DUs (via the F1 interface) on the RAN side (the suffixes "-U" and "-C" in the interface names distinguish the user

[4]In the figure we have also included the additional O-RAN components such as SMO (service and management orchestration), non-real-time RAN intelligent controller (non-RT RIC), and near-real-time RIC (nRT RIC). Since these components are irrelevant to this thesis, so we will not elaborate here.

plane and control plane versions of the standardized interfaces). Hence it may become a bottleneck in processing the downlink and uplink traffic between UPFs and DUs. We note that both the 3GPP/O-RAN NG-U and F1-U interfaces are implemented using the GTP (GPRS Tunnelling Protocol [21]) tunnels, more specifically, GTP-U tunnels, which run on top of UDP/IP over Ethernet. Hence CU-UP can be implemented entirely using commodity servers with conventional Ethernet-based network interfaces (NICs) (and possibly also Ethernet-based smartNICs). In contrast, while DU connects to CU via the F1 interface, the connection between DU and RU requires a specialized *radio fronthaul* interface, the extended Common Public Radio Interface (eCPRI) [22].

While we can incorporate eBPF+XDP to optimize the packet processing in DU for its F1-U interface with CU-UP, in this thesis we will focus on CU-UP due to its critical role in the user plane data path between DU and UPF. The main SDAP function in CU-UP involves adding or removing QFIs (quality-of-service flow identifiers) for downlink data packets from UPF to DU or uplink data packets from DU to UPF, based on (pre-defined) user data's QCI (QoS class identifier) profiles (QCI tables). The PDCP-U functions are more involved: besides integrity protection and ciphering, the PDCP-U layer is also responsible for reliable data transfer by adding sequencing numbers, buffering data, and performing retransmissions if needed. After adding/removing the SDAP and PDCP headers, CU-UP routes the user data packets using appropriate GTP-U tunnels to DUs/UPFs.

2.5 IoT System Architecture

Here, we briefly describe IoT system architecture, IoT gateway architecture and interoperability. The IoT System Architecture comprises the following components as shown in Fig. 2.7 in any CSPs [23]. It consist of IoT devices, IoT gateway and the IoT hub. Some of the IoT Ecosystem doesn't have a IoT gateway and it depends on the setup. In the further subsection, we will discuss in detail about the purpose of using an IoT gateway.

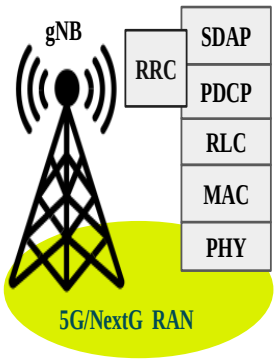


Figure 2.5: Monolithic RAN

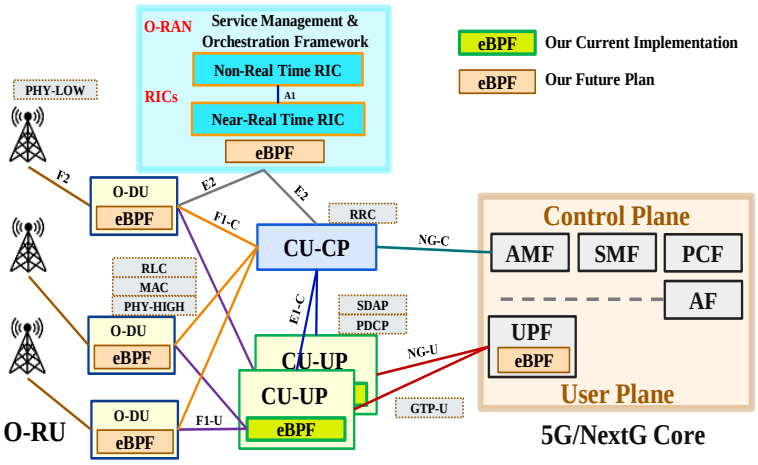


Figure 2.6: **O**-RAN Disaggregated RAN Architecture and 5G Core

2.5.1 With IoT gateway

If the building has a IoT gateway, then the IoT gateway acts as a one point communication to the internet [24]. There doesn't need to be several individual connections to the IoT hub. And also the gateway can act as a storage for the data from all the IoT devices in the network and the processing of all data can be done in the IoT gateway itself which helps certain local decisions to be performed for all the IoT devices at one location.

2.5.2 Without a IoT gateway

If the building has multiple IoT edge devices without a IoT gateway, then each IoT edge will establish its own tunnel to the IoT hub and each device needs to have its own security credentials. Each IoT edge needs to have its own storage and each device has to process its own data and certain decisions involving multiple devices cannot be performed at one location locally. And the decisions which involve multiple IoT devices need to be performed in the cloud which increases the latency in taking decisions.

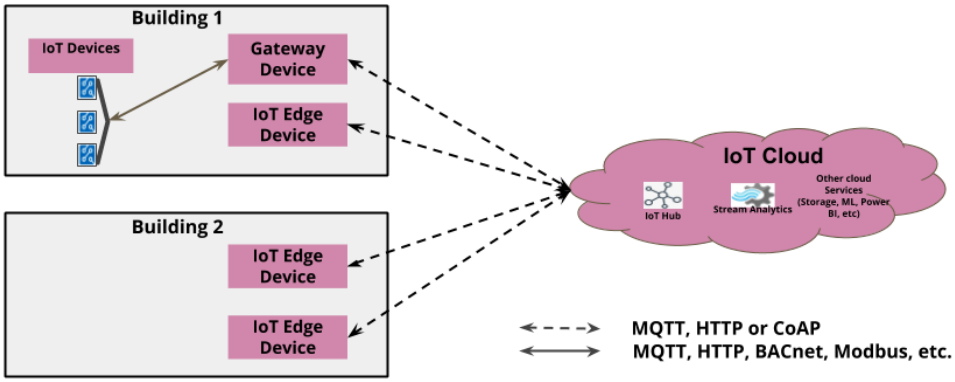


Figure 2.7: IoT System Architecture

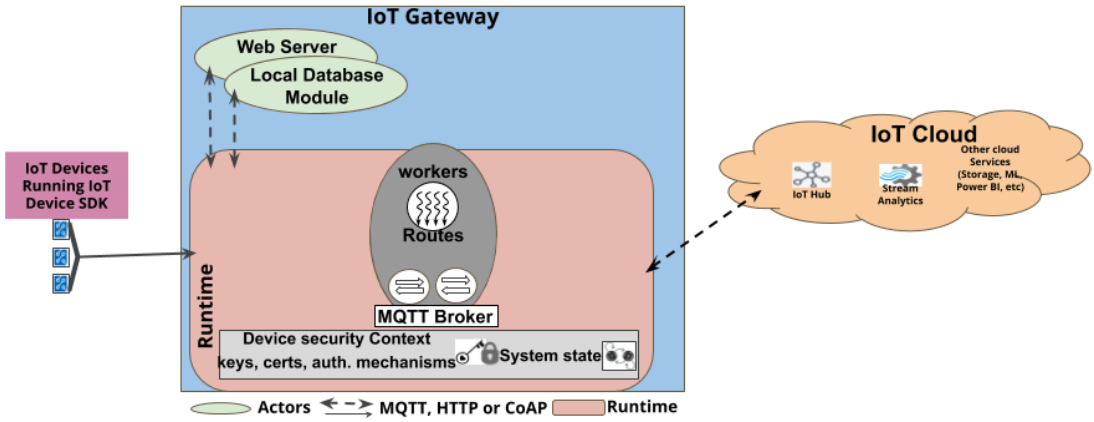


Figure 2.8: IoT Gateway Architecture

2.5.3 IoT Gateway Architecture

There is a mix and match of components in the IoT gateway architecture depending on the vendor. Both AWS and Azure have few things in common and Google is still catching up with the other 2 competitors [23]. A high-level IoT gateway architecture is shown in Fig. 2.8. Some of the common components include MQTT broker, runtime, containerization of modules, message subscriptions and connectivity to the respective IoT hub.

2.5.4 Interoperability

Interoperability in broader terms is defined by IEEE as *"The ability of two or more systems or components from different manufacturers to communicate and exchange data and to mutually use the information that has been exchanged"*[25]. More specifically, IoT Interoperability according to [26] is defined as the ability of two systems to communicate and share services with each other. The ability for two or more systems to interoperate is classified in four different layered models according to [4]: 1) *Technical interoperability:* Connect heterogeneous IoT devices at a technical level (Device level or Network Stack), 2) *Syntactic interoperability:* Interoperate the format and data structures used in the exchanged of information amongst heterogeneous IoT system entities, 3) *Semantic interoperability:* Connect the data exchanged and knowledge from IoT system components in a meaningful way, on and off the Web and 4) *Platform interoperability:* Design architectures of IoT Gateways and middleware to enable interoperability.

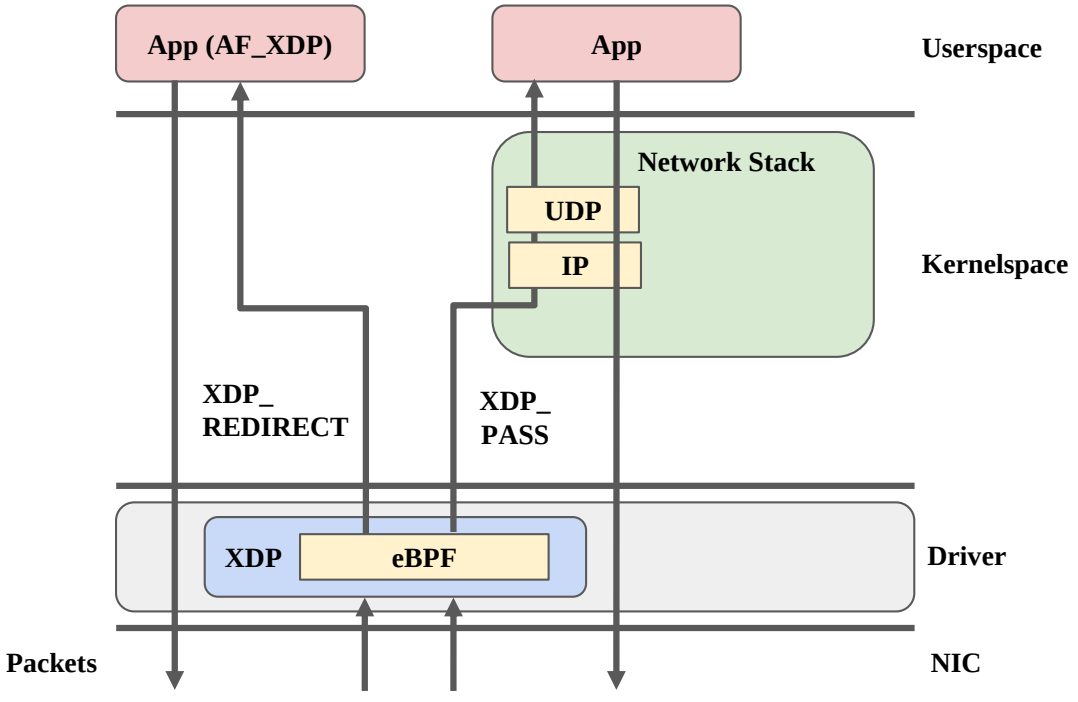


Figure 2.9: eBPF/XDP Sockets

It is important to mention that the research in this work is to investigate the design

of an IoT gateway architecture to enable seamless cross-platform interoperability via a dynamic configurable payload-based route *edge function*. We take advantage of various platforms' exposed interfaces to enable interaction between them through our proposed framework. Our architecture has three key features: 1) It can be easily integrated to other vendor platforms, 2) It enables platform interoperability irrespective of the underlying vendor-locked technologies and 3) Does not require CSPs to make major changes in their current IoT systems.

2.6 eBPF/XDP

Extended Berkeley Packet Filter (eBPF) [27] and eXpress Data Path (XDP) [28] are (relatively) recent innovations in the Linux kernel that allow safe kernel extension and kernel bypassing for more efficient network processing, among other usages. While eBPF can be used for both the transmit and receive side operations, XDP operates only at the receive side, residing within the NIC driver (see Fig. 2.9 for an illustration). We assume the reader has some familiarity with eBPF & XDP, thus will not elaborate further.

Chapter 3

VeerEdge: Towards an Edge-Centric IoT Gateway

3.1 Introduction

Many Internet of Things (IoT) systems are "stovepipe systems": they are closed, end-to-end, sensor-device-to-cloud-application systems that operate independently of each other. These stovepipe systems are unable to interact directly with each other, or share most resources. In this chapter, we describe a new class of IoT gateways that is intended to break down the barriers between these stovepipes, thereby permitting these systems to share resources, particularly the applications that process and store the sensor data. As a result, these new IoT gateways proposed here will permit the processing of sensor data to be consolidated and optionally distributed or moved closer to the IoT devices themselves.

A common example of these stovepipe IoT systems, and the challenges that these systems often present, is a home that contains IoT devices from multiple vendors, such as smart speakers. For instance, a homeowner might connect both a Google Nest Mini and an Amazon Echo Dot to their home network. Each of these IoT devices is a closed system, and is unable to share resources, subsystems, or procedures with devices from other vendors. Each IoT device is managed by its own smartphone application and communicates with its own cloud-based application. The user must learn and use two different applications to configure and manage the smart speakers. More importantly,

because the data from the smart speakers is forwarded to their respective applications, it is difficult for a single application to process the consolidated data from all of the devices simultaneously, perhaps correlating or fusing data from these separate devices. Likewise, processing the data from the smart speakers locally, or at the "edge" of the home network, is extremely difficult, because the data largely resides in the vendors' cloud-based applications.

Several major cloud service providers (CSPs), including Amazon, Microsoft, and Google, have made available IoT gateway frameworks, or software development kits (SDKs), that simplify the development of IoT devices. IoT device vendors can use these CSP-provided SDKs to simplify the development of their IoT systems: the SDKs can be used as a platform upon which to develop software that connects the vendor's IoT device to a cloud-based application. Unfortunately, each of these CSP-provided SDKs connect only to the respective vendor's cloud service. For our purposes, we call these CSP-provided IoT gateway platforms as *cloud-centric*, inasmuch as they connect IoT devices *only* to the cloud services of that CSP.

We propose an *edge-centric* model for developing IoT gateways. Instead of merely connecting IoT devices to cloud services, our *edge-centric* IoT gateway framework is designed to i) leverage computing and storage capabilities at the network *edge* (e.g., a Raspberry Pi device or a PC server collocated at a home Internet gateway or wireless base station) for edge-based device and IoT service management (e.g., fault detection, dynamic service subscription), data processing (e.g., data filtering & aggregation), and so forth; and ii) exploit availability of multiple cloud services (from different vendors) for "best" (e.g., fastest or cheapest) IoT data analytics. We summarize the outline and major contributions below.

- (§3.2) We study three leading CSPs IoT solutions to ensure our proposed framework augments current IoT gateway solutions . We especially evaluate similarities and differences between them to identify north-bound (cloud facing) and south-bound (on-premise IoT gateway facing) interfaces that can be leveraged within our proposed gateway framework.

- (§3.4) We propose *VeerEdge* - an edge-centric IoT gateway framework, that exploits the availability of multiple cloud services, storage and computing capabilities

on the network for edge-based device management and configurations and "best" IoT data analytics.

- (§3.5) We investigate a critical IoT gateway functionality dubbed - *Regulator* that realizes the edge-centric gateway vision by controlling the communication between vendor-specific IoT gateways and their respective cloud services. Proof-of-concept prototype using Amazon AWS and Microsoft Azure as case studies show that (*VeerEdge)* incurs additional negligible overhead and minimal latency.

Table 3.1: Features Supported in Current Vendor-locked IoT Gateways

Features	**AWS**	**Azure**	**Google**
Protocols	Modified Paho MQTT 3.1.1 (QoS 0 & QoS) 1 HTTP[S][29]	MQTT 3.1.1 HTTP[S] 1.1 over TLS 1.2 AMQP	Paho MQTT 3.1.1 QoS 0 & QoS 1 HTTP[S] [30, 31, 32]
Security	X.509 CA Signed X.509 Self-Signed certificates [29]	X.509 CA Signed X.509 Self-Signed certificates Symmetric keys	JSON Web Tokens [33]
Containerization Support	✓	✓	×
Message Subscriptions	✓	✓	×
Stream Manager	✓	×	×
Device Twins/Shadow	✓	✓	×
On-demand Containerization [34]	✓	×	×
Device Monitoring	×	✓	×

3.2 Terminology and Background

In this section, we use the term "IoT edge device", or simply "IoT device", to describe the end nodes in IoT systems, specifically the components that include sensors or actuators. These are the devices that generate IoT sensor data, or make changes in the physical world in response to commands. For the purposes of this chapter, we classify IoT devices into two categories:

1. "cloud-native" devices: IoT devices that are able to connect directly to applications running on a cloud service using media such as Wi-Fi or cellular service [1]. Typically, cloud-native devices are "locked" to a specific application running on a particular cloud service.

[1]While we often describe applications as running on a cloud service, they could, in fact, be running on any sort of server.

2. "gateway-assisted" devices: IoT devices that are incapable of connecting directly to an application running on a cloud service, and therefore require the services of an IoT gateway to forward sensor data to a remote application for processing. Gateway-assisted IoT devices usually connect to an IoT gateway via a low-power wireless medium such as Bluetooth, Zigbee, Z-Wave, Thread, or similar protocol because they lack the functionality to necessary communicate directly with cloud-based applications. Examples of these gateway-assisted devices include: door or window sensors, temperature sensors, or water sensors.

The data generated by the IoT devices is communicated to its end users using a pub-sub system. A pub-sub system is an asynchronous way of communicating between entities where a subscriber of a topic receives all the messages published to that topic. Amazon refers their IoT pub-sub system as *message subscriptions* [35] while Azure names it as *routes* [36]. In this chapter, we will consistently use the term *message subscriptions* or *paths* irrespective of the vendors. There are three fields required for a message subscription [35]. First, the *source*, from where the message originated. Next, the *destination*, to which the message needs to be sent. And finally, the *topic* to which one can subscribe to or publish message to.

"IoT portal/cloud" is the entry-point on the cloud. It is tied to and authenticates only specific vendors' devices and gateways and is largely responsible for heavy analysis(computation), deployment, notifications and updates. IoT gateways manage and provide connectivity to cloud-based applications for gateway-assisted IoT devices. Common consumer-grade examples of IoT gateways include smart home gateways or home automation gateways, such as the Samsung SmartThings hub. These devices implement the protocols necessary to communicate with a cloud service and applications running on that cloud service. Typically, IoT gateways implement a light-weight messaging protocol, such as MQTT or CoAP, which manages the transfer of sensor data and commands between the IoT device and a cloud-based application.

An "Edge function" run as containers [37] in the IoT gateway which is managed by the IoT gateway's run-time. Container is a unit of software that contains code and all its dependencies (run-time, system tools, system libraries and settings) as a single package. In Azure IoT, the containers packaged with custom code are called modules. And in AWS IoT, these containers are called as lambda functions [34]. Additionally, in AWS,

the lambda functions can run as an individual process in the IoT gateway instead of a container. As shown in Fig. 3.2, local database, web-server, machine learning services are some of the examples for a edge function.

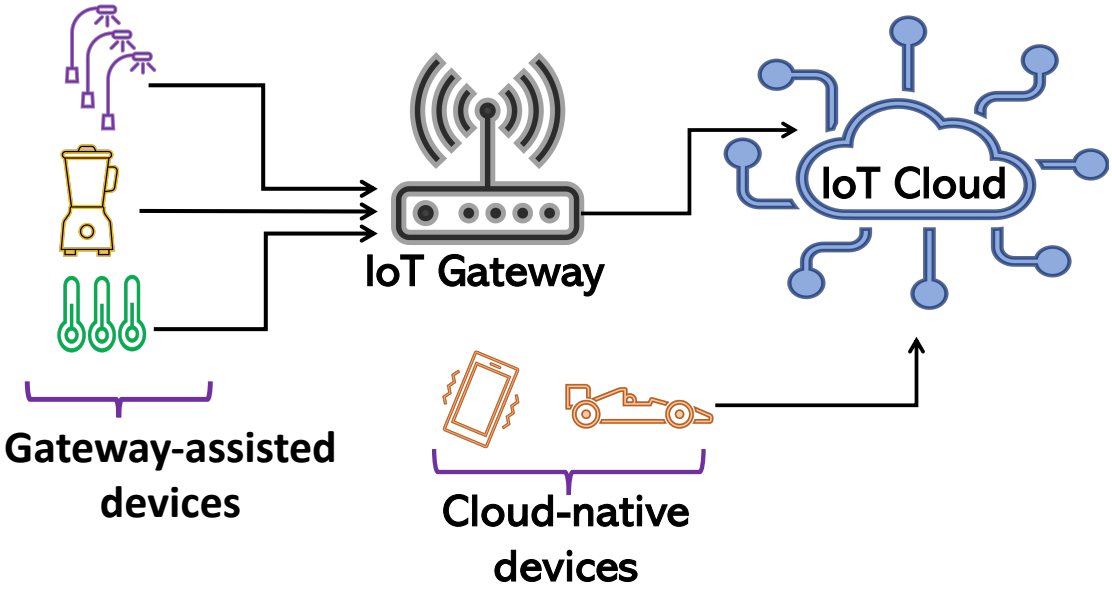


Figure 3.1: Cloud-native and gateway-assisted IoT devices

Major cloud service providers offer what we refer to as "IoT cloud services", specialized services that support IoT devices and IoT gateways. For example, these IoT cloud services generally support one or more common IoT messaging protocols, such as MQTT or CoAP. While terminology differs among cloud vendors, we refer to these IoT-specific services as "IoT portals". Cloud-hosted applications process and store IoT sensor data, initiate notifications in response to IoT sensor data, and manage IoT devices and users.

3.2.1 Cloud-Centric IoT Gateways

Recently, several CSPs, including Google [30], Amazon Web Services (AWS) [38] and Azure [24] have made available IoT gateway platforms, or SDKs, that IoT vendors may integrate into their IoT devices or gateways. These CSP-provided SDKs simplify the development of IoT devices and gateways, but at the expense of locking the vendors into the SDKs' respective cloud services. We summarize their similarities and differences in Table 3.1 and briefly discuss them here.

AWS IoT Core

Amazon's IoT Gateway SDK is called "Greengrass" (GG) [39]. One major feature of Greengrass is the runtime. GG's runtime serves both as a client to the AWS cloud and a server to "gateway-assisted" AWS devices to marshal data and enable bi-directional communication between these entities. GG's runtime is equipped with a modified Paho-based MQTT 3.1.1 implementation over TLS 1.2 encryption (MQTT over Websocket) with X.509 certificate-based mutual authentication [29] [2]. The runtime also offers support for Lambda functions – a server-less compute service to run code in response to an event [38]. This functionality may simplify events-response and control within IoT applications. The GG's runtime maintains IoT device state information using "Device shadows" via a JSON serialization format [29] which simplifies management for mobility support. Additionally, the runtime performs data aggregation, queuing and scheduling before forwarding IoT data to the cloud.

Microsoft Azure IoT

Microsoft Azure IoT gateway SDK is also equipped with a customized runtime. At this time, the runtime relies on a broker to communicate with the cloud. This broker can be configured with either MQTT 3.1.1, AMQP or HTTP 1.1 protocols secured with TLS 1.2. and token-based authentication [40]. Azure provides a simplified version of Lambda functions called "modules" – to run code based on a trigger [36].

Mobility support is provided via "device twins" – a customized but different [3] JSON serialization format [41] to maintain IoT device state information.

Google IoT Core

Googles' gateway SDK is an embedded-device SDK. It supports HTTP 1.1 or a custom Paho-based MQTT 3.1.1 protocol with TLS 1.2 with JSON Web Tokens (JWTs) [42] for authentication [30, 31, 32]. Google's gateway does not have a runtime. However, device state information is maintained using "device metadata" (maximum size of 256 KB),

[2] At the time of writing this thesis, GG V1 included MQTT QoS 0, "fire and forgot" which does not require any acknowledgement and QoS 1, "fire and wait for acknowledgement" ensures an acknowledgement is received.

[3] When compared with AWS device shadows.

"device configuration" (maximum size of 64 KB) and "device state" (maximum size of 64 KB) [43]. Unlike Azure and AWS that only support JSON serialization format, Google also supports binary, text or serialized protocol buffers data formats [43].

We acknowledge that these vendor gateway SDKs render significant contributions within the IoT systems. Nonetheless, they all embrace a cloud-centric approach and still posses major drawbacks. Next, we describe their drawbacks and then make a case for an edge-centric approach for IoT gateways.

3.3 Challenges with cloud-centric IoT gateways

Due to the recent surge of IoT devices, several IoT applications have been deployed in the cloud to perform computation on the IoT data. Limitations like latency perceived by end-systems, and increased bandwidth usage between the end-systems and the cloud ought to move IoT data computation towards the edge. However, key-players like Microsoft, Amazon, and Google only allow the management of IoT data on the cloud. This in turn makes the management of IoT data that utilizes cloud-based applications from different vendors, cumbersome. In this section, we explain in detail these challenges posed by cloud-centric IoT gateways. As a results, presents a unique opportunity to advocate an edge-centric IoT gateway in order to unlock the benefits of various IoT applications provided by different vendors and to also enable convenient management of IoT data.

3.3.1 Cloud-Centric IoT Gateways: Issues

To marshal IoT data in current industrial IoT solutions, *message subscriptions* needs to be configured in the cloud and then deployed in the IoT gateways. The data from IoT devices is then routed through the IoT gateway to the relevant users based on these subscriptions. This, however, poses several challenges in developing innovative and rich IoT solutions, as discussed next.

Cumbersome cloud-based IoT device configuration. In-order to disable or enable communication of IoT devices with the cloud, we need to completely remove or re-add the paths that exist in the cloud and redeploy them on the IoT gateway. In AWS and Azure, the paths need to be configured to enable communication between IoT

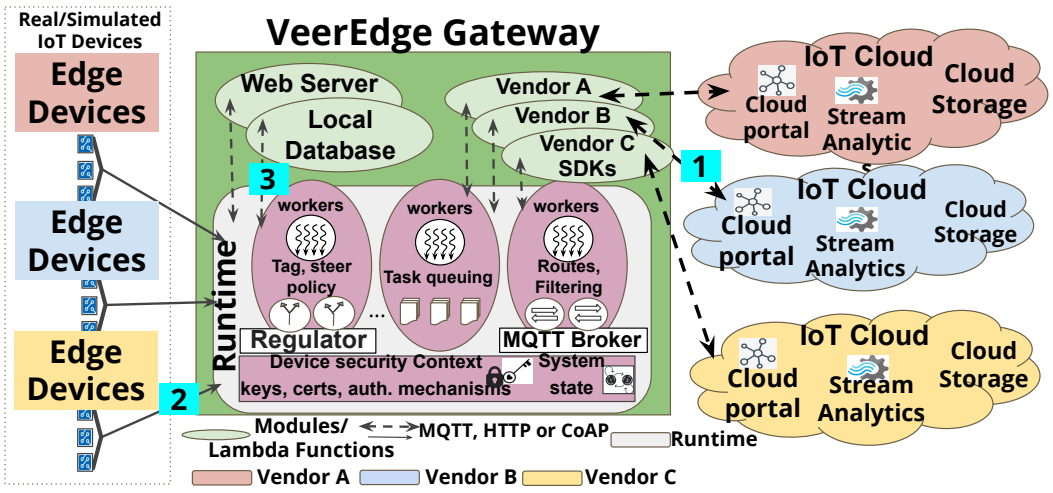


Figure 3.2: Edge-Centric IoT Gateway Framework

devices, edge functions and the IoT cloud.

Inflexible cloud-only IoT data management. Cloud-only management of IoT data makes its management very difficult. To utilize applications from different CSPs' clouds, we need multiple IoT gateways and also need to configure several paths on each cloud portal. Such configurations in the cloud portal incurs higher latency since the cloud is generally far away from the end-users. Moreover, these paths have to be deployed in the IoT gateway after re-configuration in the cloud, thus, adding up to this latency. For example, if a building already have a Azure IoT gateway, in order to send videos data from an IP camera to AWS Kinetic Streams, an AWS GG IoT gateway need to be installed and configured in the building. Management of multiple IoT gateways is time consuming and challenging due to maintainability.

No/little support for cross-vendor edge computation and data analytics. Vendor IoT gateways use the paths deployed in it to simply forward the IoT data to vendor-specific applications hosted in the cloud. The inability to configure these paths at the edge prevents IoT data to be dynamically routed to other CSPs' cloud or edge. Leveraging "better" stream analytics and machine learning application of another vendor is extremely difficult and impractical.

3.3.2 Case for an edge-centric IoT gateway

The challenges identified in the previous subsection leads us to advocate an *edge-centric* architecture for designing IoT gateways. Instead of merely connecting IoT devices to cloud services, we envision an *edge-centric* IoT gateway that i) leverages computing and storage capabilities at the network for edge-based device management, configuration and control, and ii) exploits the availability of multiple cloud services (from different vendors) for "best" (e.g., fastest or cheapest) IoT data analytic. An *edge-centric* IoT gateway ought to enable sending IoT data to clouds of different vendors instead of locking it to a specific vendor. Moreover, Edge-centric IoT gateways also make IoT applications less prone to security attacks since there is more privacy compared to in the cloud as the data resides locally. We achieve this by introducing *regulator*. *Regulator* is a subsystem built atop existing vendor IoT gateway SDKs and enables flexible device configuration and data management, dynamic cloud service subscriptions and message routing. We provide a detailed description of *regulator* later.

3.4 How to address these challenges?

Given the challenges mentioned earlier, in this section, we discuss various solution approaches and highlight their limitations. We, then present our *VeerEdge* IoT gateway architecture design.

3.4.1 Re-designing the IoT Gateway

One approach to address these challenges might be to re-design an IoT gateway from scratch. This gateway should be open and not tied to any specific CSP IoT cloud portal. Rather, it should provide multi-cloud support to connect, communicate and exchange data with several vendor IoT cloud portals while still enabling local configurations. This approach replaces the current CSPs' IoT gateway solutions and imposes a unified ontology or a consensus of the communication protocols and RESTFul APIs adopted. According to the European project Unify-IoT, more than 300 IoT cloud platforms exist today [44]. Therefore, an obvious problem with this approach is its inability to work with existing IoT cloud portals, *i.e.*, CSPs may need to re-design their IoT cloud portals

to add support for the unified APIs and protocols. This is impractical, time consuming and might be less beneficial for some vendors.

Thus, several vendors might be reluctant to proceed with an agreement.

3.4.2 Building atop current IoT solutions

In this study, we take a different approach. Instead of re-designing an IoT gateway from scratch, we build atop existing IoT gateways and only leverage their runtimes. We propose a wrapper dubbed, *Regulator* which leverages vendor gateway SDKs to connect to their respective IoT cloud portals. Specifically, this approach augments current systems' runtimes and take control of all communication happening between the different vendor IoT gateways and their clouds portals.

The limitations with our approach are two folds. 1) There is no unified way for the IoT devices to communication with the IoT gateway. Each vendor will still advocate its unique security mechanisms for IoT device to authenticate with the IoT gateway. 2) This solution is limited to "gateway-assisted" IoT devices. Nonetheless, in this study, we investigate this approach and augment current IoT gateway SDKs enabling multi-cloud support, edge-computation, dynamic manageability using payload information within *Regulator*, while still using the cloud.

3.4.3 Proposed VeerEdge Gateway Design

Fig. 3.2 summarizes the detailed architecture of *VeerEdge*. We discuss the major components of *VeerEdge* below.

Runtime

In our design, the runtime does the heavy work. We build atop existing IoT gateways by leveraging existing vendor gateway runtimes. AWS GG and Azure IoT gateway runtimes come pre-build with a task scheduler, and an MQTT Broker. We therefore use both runtimes during our implementation and show evaluation results in §3.5. We augment their runtimes with *Regulator* - edge function.

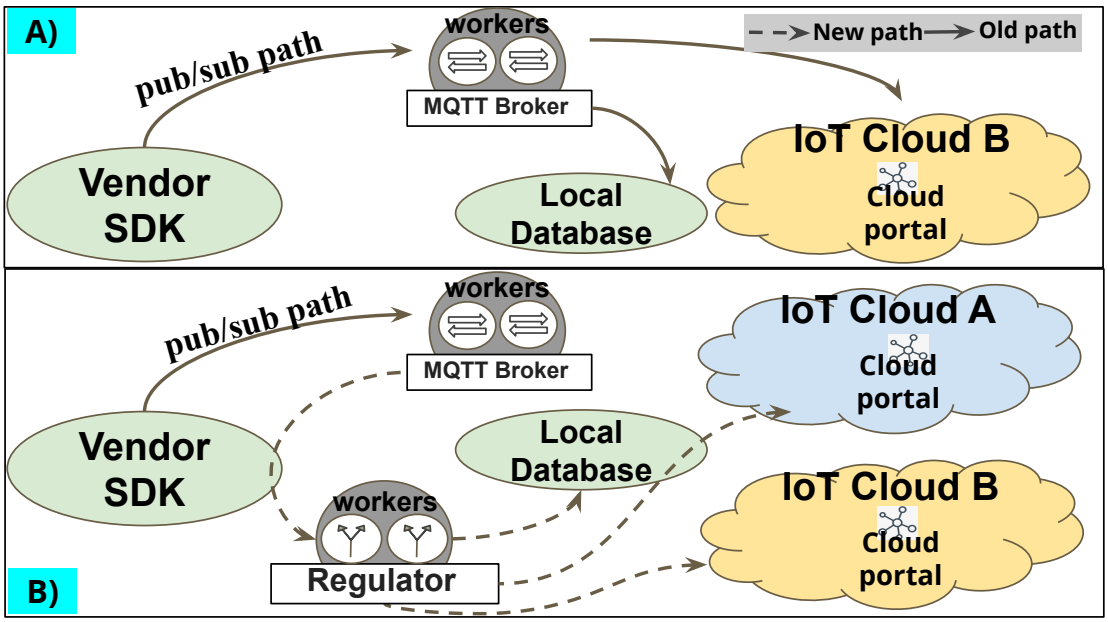


Figure 3.3: *Regulator* operation scenario.

IoT gateway SDKs

Within our edge-centric IoT gateway, we run Google, Azure and AWS gateway SDKs [45, 24, 39] as edge functions. They provide the RESTFul APIs to communicate with their IoT cloud portal. However, as describe later in sec. 3.4.3, *Regulator* controls all paths in our design bringing IoT data closer to the edge. We host a local database for data storage and a web server for local configurations and management, alleviating the cloud-centric management.

Regulator

Regulator is primarily controlled by user configuration via the local webserver. Specific configuration options like "disable publish to cloud", "delete path x" and "create path y" can be configured. We consider "disable publish to cloud" in §3.5 within *regulator* during our implementation. This is because, our approach here involves creating and deleting (temporarily disabling) paths. Traditionally, performing this function ("disable publish to cloud") requires, manually deleting and re-deploying the configuration locally on the gateway on premise. In *VeerEdge*, the webserver performs a runtime interrupt via

regulator. (i) *Regulator* leverages the runtimes' exposed APIs to discover the current static paths (source, destination, topics) pairs. (ii) It creates (if it does not exist) a new "shared topic" and subscriber (usually the local database) on the system and (iii) temporally disables the cloud facing path and redirects every packet via the new "topic" (path). This simple operation is summarized in Fig. 3.3.

By means of this functionality, *Regulator* can, 1) dynamically operate on all vendor IoT Platform SDKs edge functions deployed on the system, 2) avoid downtime that exist when re-deploying new cloud configurations and 3) seamlessly reduce unconnected "stovepipes" between vendors, thus interoperability. However, for the first time, the path need to be configured in the IoT cloud and deployed to the IoT gateway. And the *regulator* controls the paths thereafter.

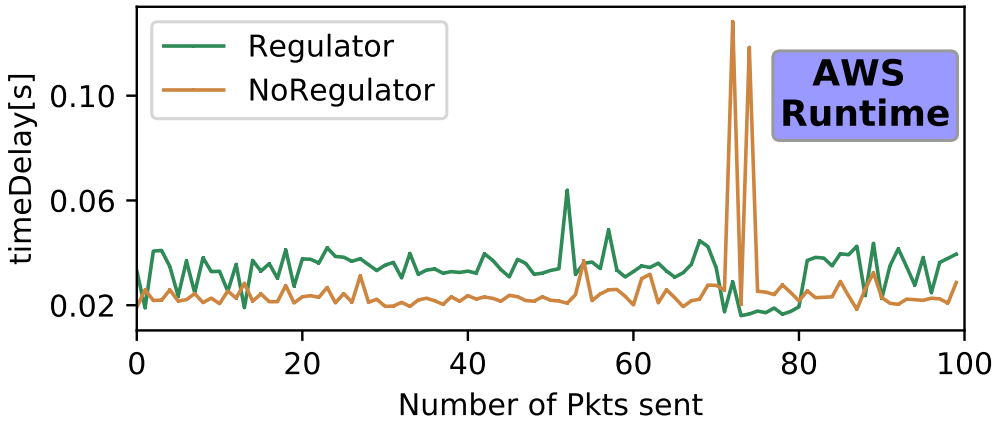


Figure 3.4: AWS runtime time delay - Jetson Nano

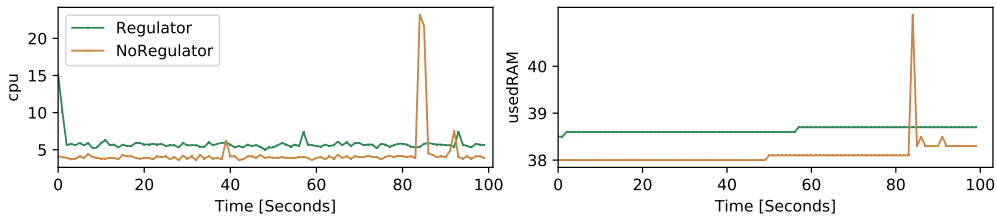


Figure 3.5: AWS runtime memory and CPU utilization - Jetson Nano

3.5 Implementation and Evaluation

In this section, we discuss the implementation and evaluation results of our proposed design.

3.5.1 Implementation

We first implemented *VeerEdge* on a Raspberry Pi2 and show preliminary implementation results in [46]. Next, we extended our evaluations and implemented *VeerEdge* on an NVIDIA Jetson Nano. The web-server is used for local configuration of paths, the local database stored IoT data locally and Vendor A, B and C SDKs are Google's, Azure's and AWS' Gateway SDKs equipped with all security context for authenticating and communicating with their clouds portals. *Regulator* leverages these SDKs to control and steer traffic based on the paths configured. At start-up, *regulator* takes over the paths and controls the traffic based on the local user configuration. Through this implementation, we were able to enhance the existing IoT gateway frameworks to support local enable or disable of the message subscriptions without using the cloud portal. Since the paths already exists, the communication between two entities pass through without any issues and our implementation controls the pass-through only to disable or alter the communication. Through this approach, we have eliminated cloud-based configuration and an additional deployment. Additionally, *regulator* enables the local control of paths between vendor-specific IoT gateway and multi-vendor IoT clouds, which is not supported with current vendor Gateways.

3.5.2 Evaluation

First, we seek to quantify the additional delay incurred by processing every packets via *regulator*. We simulated IoT devices to publish data to our universal gateway and logged the time when every packet was sent. We configured a path to route every packet to the local database, where we logged the time every packet was received. We repeated this experience with and without *regulator*. In Fig. 3.4, we show the additional delay incurred with *regulator* (the blue curve) and without *regulator* (the red curve) leveraging the AWS GG runtime. Noticed that, *regulator* incurs negligible overhead while addressing interoperability challenges in current CSPs' IoT platforms.

Next, we collect the CPU and memory utilization with and without *regulator* to understanding the additional overhead incurred with our approach. As shown in Fig 3.5, augmenting both AWS GG runtime with regulator result in more CPU and memory usage as expected. This is because, all packets through the gateway are processed by regulator, i.e., the payload needs to be matched to paths deployed on the system before routing. These results show this approach incurs additional negligible overhead and minimal latency. It is important to mention that, we acknowledge that, the additional overhead incurred by this approach can be problematic in low latency IoT application scenarios. Nonetheless, the results are promising.

3.6 Summary

In this study, we make the case to advocate a shift from a cloud-centric to an edge-centric approach in IoT gateways. We proposed *regulator*, which augments current vendor-locked IoT platform solutions by controlling paths from various vendor gateway SDKs to the cloud. We evaluated our methods by using both AWS and Azure runtimes. Our experiments show that our approach incurs negligible overhead and minimal latency. Although we developed an approach to support interoperability from the IoT gateway to the cloud, unlocking multi-vendor downstream IoT devices to vendor IoT gateway still remains a challenge, especially since vendors adopt custom security mechanisms in their IoT devices, gate SDKs and IoT cloud. Moreover, there is no standard message subscriptions framework followed by the CSPs: AWS uses MQTT topics and Azure uses endpoints. This challenge is left for future works. Our experiments clearly show that the resource consumed by the Azure IoT gateway framework is relatively higher than the AWS Greengrass. Thus, further exploration of both frameworks to understand why can be a possible research direction. In summary, suggested future research direction can be; 1) addressing the interoperability issues with vendor IoT edge devices and 2) building an IoT gateway runtime that supports heavy edge computational tasks and connects to multiple vendor cloud platforms.

Chapter 4

Kaala 2.0: Scalable IoT/NextG System Simulator

4.1 Introduction

The proliferation of IoT devices in recent years has made it possible to develop innovative smart services for homes, offices, businesses, cities and communities. Large cloud service providers, such as Amazon, Microsoft and Google, offer cloud-based IoT data analytics and AI services for collecting, storing and processing the massive amounts of IoT data these devices generate. Because these new cloud IoT services often obviate the need for IoT device vendors to deploy their own data centers, cloud services have become integral components of most IoT systems.

Cloud IoT service vendors, such as AWS [38], Azure [47], Google [30] and Alibaba, all aim to build their own IoT ecosystems, which currently do not interoperate with each other. According to the UNIFY IoT project, more than 360 IoT companies exist today [44]. While there are industry-led efforts to ensure interoperability between cloud IoT services (e.g. via CHIP [48]), non-interoperable cloud IoT services are likely to remain the rule, rather than the exception, for some time.

An alternative to waiting until physical devices have been developed and constructed, is to use IoT simulators to test and evaluate prospective IoT devices, systems, and designs. If these simulators and experiments are designed properly, simulation can significantly reduce the gap between proof-of-concept (PoC) implementations and real

world deployments [49]. Unfortunately existing IoT simulators are limited in their capabilities and scopes. 1) Many simulators are designed to run on a single laptop, desktop or a server, and are therefore poorly positioned for large-scale simulations that require significant computational power. 2) Most of simulators fail to capture the large variety and diversity of IoT devices that exist today. For example, many are tailored to simulating only small sensors with low bandwidth requirements, ignoring a variety of IoT devices (e.g., surveillance cameras and autonomous vehicles) that consume large amounts of network bandwidth and require real-time cloud connectivity. 3) Perhaps more importantly, existing IoT simulators operate in isolation: they interface with only a limited number of types of IoT devices and can not be integrated with existing cloud services [50]. In short, existing IoT simulators cannot be used to effectively test and evaluate prototype IoT systems, especially those that require computationally intensive subsystems such as machine learning algorithms, Cloud IoT services, or IoT control mechanisms running on edge computing facilities. Chernyshev et al. [49] in a recent survey highlighted that an all-in-one simulator capable of supporting an end-to-end IoT service is yet to be developed. Simulation tools and environments for assisting testing and evaluating of unit testing and systematic evaluation of a smart IoT services with diverse devices, edge and cloud computing components in an integrated fashion are therefore sorely needed.

In this chapter, we present *Kaala 2.0* – a modeling, simulation and emulation platform that is capable of specifying IoT devices of various types, from low-powered sensors to smart IoT devices requiring high bandwidth, such as IoT devices that anticipate emerging 5G networks. Kaala 2.0 is an extension of Kaala [51] and simulates UE, RAN and 5G Core at the same time connect to real-work UE, RAN and 5G Core. In addition to simulating IoT devices, an important feature of Kaala 2.0 is its ability to interface and connect with real-world cloud IoT services in an integrated fashion. The initial version of Kaala 2.0 can use Amazon AWS [38], Microsoft Azure [47] and Google [30] IoT cloud services. The main design goal of Kaala 2.0 is to help researchers and practitioners to prototype various IoT scenarios, including those that use high-bandwidth 5G services, generate massive amounts of data, and to help bridge the gap that exists between simulators and real-world system. The major contributions are summarized below.

- We present Kaala 2.0: An IoT modelling and simulation platform that is able to specify IoT devices of various types to communicate with real-world cloud IoT systems, with AWS, Amazon and Google IoT Cloud platforms as case studies through simulated or real-world 5G networks.
- Kaala 2.0 is a scenario-based IoT simulator capable of mimicking various IoT scenarios such as "fire in a room or building" to evaluate URLLC service of 5G and "5G network capable data generation (including 4K/8K video IP cameras)" scenarios.
- Kaala 2.0 is able to simulate massive number of IoT devices to evaluate mMTC service of 5G.
- Kaala 2.0 is able to generate massive amounts of IoT data for prototyping data-intensive IoT applications to evaluate eMBB service of 5G.

The rest of the chapter is organized as follow. We motivate the design and use cases of Kaala 2.0 in § 5.2. The design and implementation of Kaala 2.0 are presented in § 4.4 and § 4.5, respectively. Kaala 2.0 is evaluated in § 4.6 and conclude the chapter in § 4.8.

4.2 Case for Kaala 2.0

In this section, we use three case studies to argue the need for a better simulation framework, while discussing their challenges.

4.2.1 Ability to Interact with Real Systems using 5G Networks

Emergent IoT applications are extremely complex and operate in a very diverse IoT world. These are not just the smart speakers, smart thermostats, smart door locks in our homes, but also sensors used in domains like in the oil, gas and automobile industries. These applications support different systems connected through IoT. Researchers study, implement and test IoT prototyping ideas on simulators. These simulators by far do not reflect the complexity that exist in the IoT world. For example, current IoT simulators do not simulate vendor-specific IoT devices [13]. The IoT simulator provided by AWS [14] can only simulate one type of IoT device. It simulates hard-coded IoT

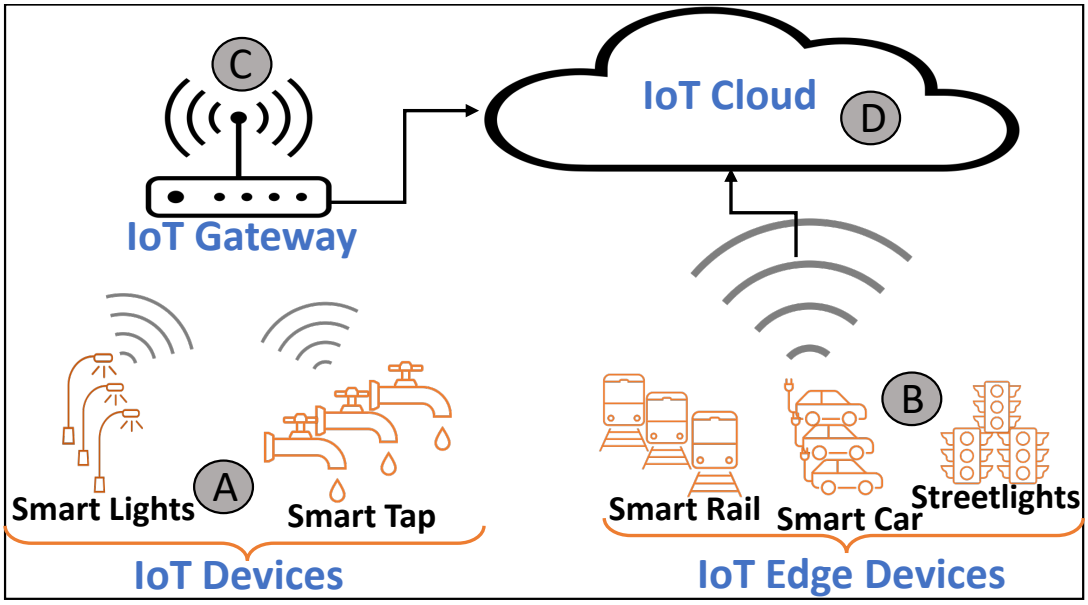


Figure 4.1: IoT (Edge) Devices, IoT Gateway and IoT Cloud

messages and does not simulate the network characteristics (TCP/IP stack) along with massive IoT data. To the best of our knowledge, current IoT simulators operate in isolation and do not interact with real cloud IoT system, failing to reflect the complexity present in the IoT world. Kaala 2.0 is intended to remedy these deficiencies. Kaala 2.0 simulates several vendor-specific (AWS, Google, and Azure) IoT devices with network characteristics capability to connect and communicate with real cloud IoT systems. Kaala 2.0 connects simulated IoT devices with real servers (within the complex IoT world), so that services provided by cloud service providers can be used, validated and verified. For instance, Kaala 2.0 connects simulated IP cameras, temperature sensors, humidity sensors, flame sensors to the Amazon's Kinetic video streams and building logic around these sensors to simulate a fire event.

4.2.2 Scenario-based Data/Event Simulation

We use a *fire-in-a-building* event to make the case for the need of a scenario-based data generation and 5G service. IoT data generated by current IoT simulators are hard-coded [13]. That is, the data generated do not realistically models IoT data generated by real sensors. A more realistic approach might be to generation sensor data based

on a distribution or based on the actual behavior of the sensor. Consider a *Fire-in-a-room* scenario. When there is a fire in the building/room, the temperature in the room increases and the temperature sensor will report a higher value than usual. The smoke sensor will detect the smoke in the room and send the smoke alarm. The humidity in the room increases and the humidity sensor will be sending the updated humidity value. Some sensors might malfunction, burn or lost connection because of the fire. Thus, relying a few sensors data values will be problematic. We might want to analyze data from some other IoT devices (like a camera) to understand the prior and current situation of the fire event to call an ambulance. IoT simulators ought to model such scenarios. The inability of current IoT simulators to model real-world IoT scenarios present Kaala 2.0 with a unique opportunity. We present more details of how Kaala 2.0 achieves this in § 4.4.2.

4.2.3 High-bandwidth Data Generation

IoT simulators ought to support next-generation network technologies such as 5G. With higher 5G throughput, next generation IoT applications should seamlessly adapt to 5G. However, this is not the case. This is due to the lack of tools (IoT Simulators) which foster the design, development and deployment of 5G capable applications both on the client and server side. For instance, a video streaming service provider like YouTube or Netflix have millions of users watching videos. The throughput will depend on the quality of the video. Recently 8K videos require significantly higher bandwidth (5G speeds) to play a single frame compared to a 480p video quality. For example, a video of 'X' minutes sizes 119 MB for a 240p video. requires 1038 MB for a 1080p video and 7284 MB for a 8K video. Thus, prototyping 5G next generation IoT applications using an IoT simulator that can support massive data workloads seen in the production environment should be addressed. Current IoT simulators do not support modelling thousands of IoT devices with large amount of data. Kaala 2.0 realises this challenge by simulating IP cameras that supports 8K video streaming and is able to scale to hundreds (and even thousands) as shown later in § 4.6.

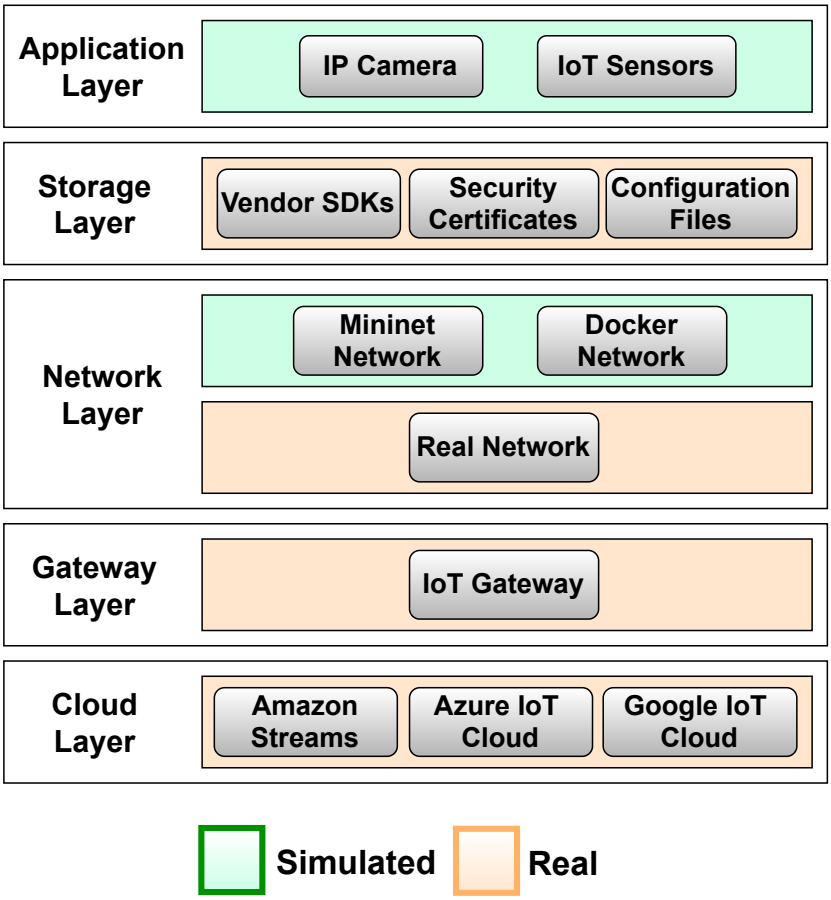


Figure 4.2: Kaala 2.0 Layered Architecture

4.3 Kaala 2.0 Architecture

Fig. 2.7 shows the system design of Kaala 2.0. As shown, Kaala 2.0 is able to integrate real and simulated devices while leveraging vendor specific SDKs to connect them to real systems in the cloud. Next, we motivate and detail each layer in Kaala 2.0's architecture as shown in Fig. 4.2.

There are three main objective of Kaala 2.0. 1) Simulates various IoT devices including vendor-specific IoT devices and connect them to particular vendor's cloud IoT services. 2) Simulates realistic data across all applicable devices to mimic real IoT service scenarios. 3) Generate high-throughput real data.

Cloud layer: This layer is the real cloud IoT systems that Kaala 2.0 connects to.

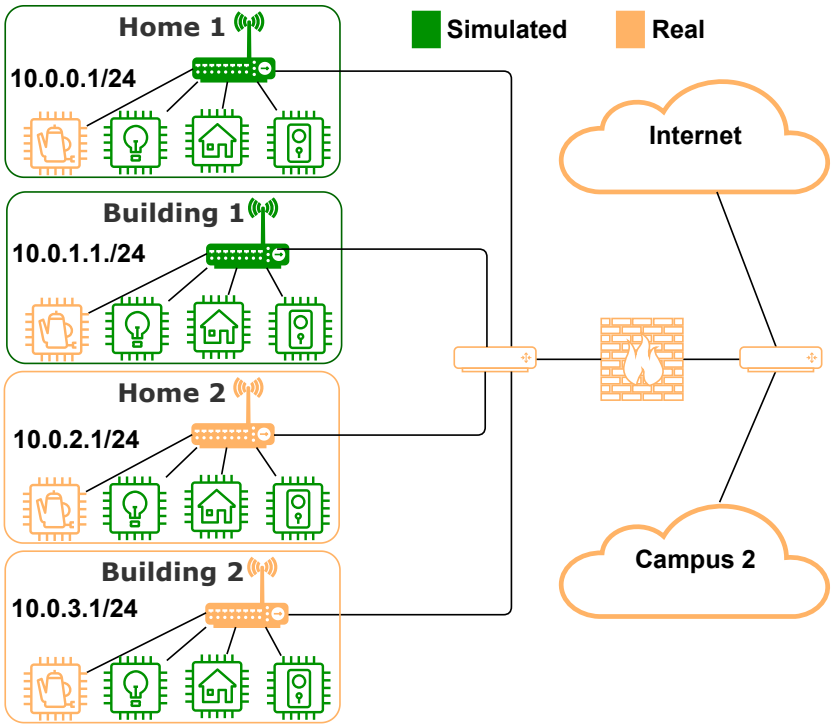


Figure 4.3: Kaala 2.0 Network Architecture

It authenticates, validates and accepts incoming connections and IoT data from the IoT gateway. The cloud layer provides core entities for massive data transformation, data analysis and interpretation. It provides data stream processing resources and cloud services for machine learning related tasks, business integration and user management. It is also responsible for notification and historic data storage.

5G Core Layer: This layer is the simulated 5G core to which each of the simulated RAN connects to. On the other end, the 5G Core connects to the internet. This can be replaced with real-world 5G core as well in which the RAN also should be real-world 5G RAN and the UE need to have proper service to the real-world 5G provider.

RAN Layer: This layer is the simulated RAN to which the simulated UE's connect to. There can be more than one RAN and each RAN connect to one 5G core. This can be replaced with real-world 5G RAN which connects to real-world 5G core as well in which the UE need to have proper service to the real-world 5G provider.

Gateway layer: The IoT gateway connects Kaala 2.0 to real cloud IoT systems.

It runs vendor-provided SDKs, which contain the RESTFul APIs needed to connect to the the cloud layer. It serves as an MQTT broker (server) to IoT (Edge) devices and is a client that connects to the cloud. The MQTT broker receives IoT data from the network layer and translates the data into vendor-specific data formats via their SDKs. It retrieves security contexts from the storage layer, authenticates, validates and connects to the cloud layer to forward the IoT data to the IoT cloud.

Network layer: The network layer connects all the other layers. It is a virtual network that models a real network. It contains core network components like a DHCP server, a DNS server, a switch and a gateway. Each modelled physical devices has a virtual IP address. This layer provides the resources needed by the gateway to connect to real systems via their domain names. Every instance of this layer creates a separate isolated virtual network that can connect to real systems as shown in Fig.4.3. Our evaluation results in § 4.6 show quantitative statistics of network utilization for an end-to-end IoT scenario experiments.

Storage layer: The storage layer comprises of different vendor SDKs, security contexts, data storage and configuration files. Cloud service providers support different SDKs, RESTFul APIs, data formats, security mechanisms, certificates and keys. This layer is responsible of contacting the service providers and obtaining the updated certificates, keys and SDKs used by the gateway layer or the application layer for authentication and validation.

Application layer: This is the layer responsible for simulation configurations, parameters tuning, IoT device configuration, network configuration and experiment scenarios. The main purpose of the application layer is to run application specific logic. The application layer provides standard IoT device functionality, such as publishing messages but the architecture supports easy extension of IoT device-specific logic by inheriting from the base IoT device logic. One example for the application logic extension to basic application in Kaala 2.0 is adding support for IP cameras in which the application creates IP camera related functionality to support high-bandwidth data which is discussed in detail in § 4.4.2 and § 4.5.

4.4 Kaala 2.0 Design

In this section, we discuss Kaala 2.0's design goals followed by a detailed design to guide our implementation. The key design goals for Kaala 2.0 are four folds: 1) Connect simulated IoT devices to real cloud IoT services, 2) provide extensibility of IoT device characteristics for current and future IoT devices, 3) simulate real-time events coordinated across multiple IoT devices and 4) generate realistic data to support current and future technologies like 5G. In the next sub sections, we will discuss how Kaala 2.0 achieves these design goals.

4.4.1 Interacting with Real-World Systems using 5G networks

To support connection with both simulated and real networks including 5G networks, we leverage Mininet [52] and docker [53]. Mininet has the capability to create various types of virtual networks using different types of switches and controllers, and each host will be running as a process. Docker has the capability to create virtual networks and each host will have its own container [53]. These Mininet and docker virtual devices have IP addresses assigned to them and therefore can connect to real physical networks. As discussed earlier in § 4.2.1, we enable interoperation with real cloud IoT systems by integrating vendors' SDKs [47]. Both the Mininet and docker architecture supports running real application processes in the respective host instances. Using this, the simulated applications run the respective SDKs in the various hosts as processes. Since the host application runs as a process, the host has access to all the files stored in the machine in which the simulation framework is running. As a result, each host does not need to have the individual vendors' SDKs installed and can reuse the SDK installed in the machine in which the simulation framework is running. This architecture is highly scalable when compared to the architecture proposed by IoTNetSim in[13]. This is because IoTNetSim creates virtual machine for each of the simulated devices. Moreover, in IoTNetSim the SDKs need to be installed individually in each virtual machines. We compare the scalability of Kaala 2.0 with IoTNetSim in § 4.6. The key advantage of using Mininet is the ability to create a wide variety of multiple network architectures based on a simple configuration [52]. Additionally, various network configurations with different type of controller and switches can also be simulated and test for IoT traffic.

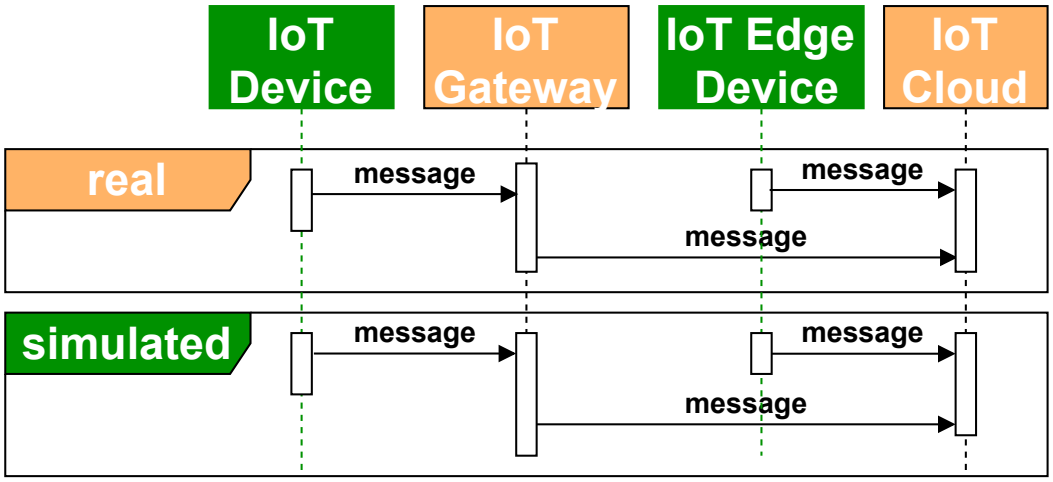


Figure 4.4: Sequence of data flow showing no differences in data flow between real and simulated IoT devices

Simulated IoT devices connect to the IoT gateway and we cover the implementation details in § 4.5. Earlier in § 5.2, we motivated the need to simulate vendor-specific IoT devices. In our proposed simulation framework design, both generic IoT devices and vendor-specific IoT devices can be simulated. We isolated each of the IoT devices with its own network resources. As a result, simulated IoT device can express the characteristics of a real IoT device and also run application specific code for the respective simulated IoT device. The simulated vendor-specific IoT device each run the respective vendor IoT device SDK to connect to the respective vendor-specific IoT gateway or IoT Clouds.

4.4.2 Scenario-based Event Simulation

The design of Kaala 2.0 supports most of the real-time scenarios. We have also discussed a couple of scenarios in §4.2.2. The basics of scenario based simulation is to coordinate one or more simulated IoT devices to match values based on the scenarios at the same duration range. Kaala 2.0 supports simulation of one or more scenarios at the same time or in sequence. The fire in the room scenario is an in-built scenario in the Kaala 2.0 simulation framework. When there is a fire detected in the room, the smoke sensor detects the smoke, the door lock opens automatically, the temperature in the room increases, the humidity in the room increases as well and the IP camera in the room

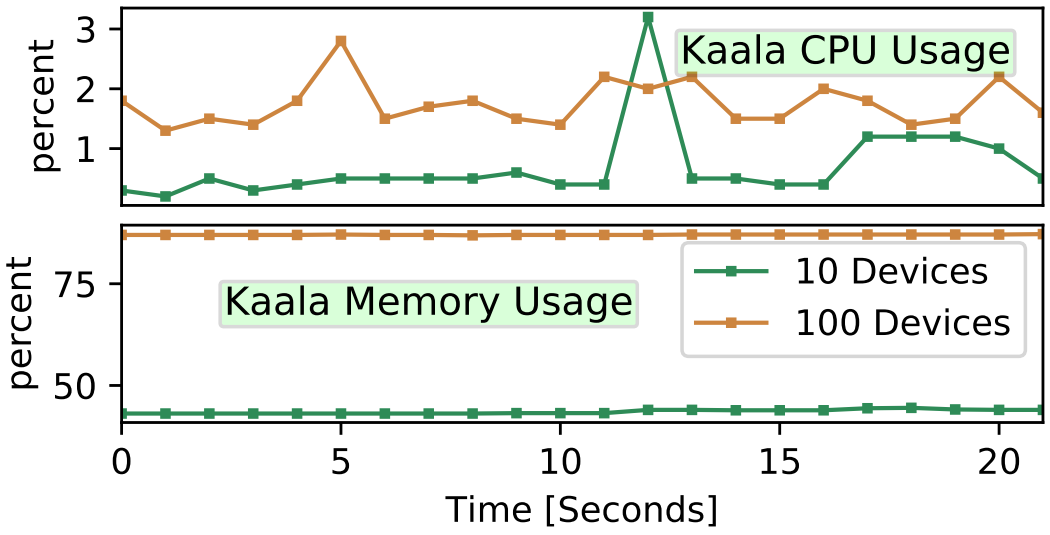


Figure 4.5: Kaala 2.0 Performance Evaluation - Mininet

captures the video. The list of affected IoT devices and the respective values need to be configured in the scenario configuration file which gets loaded while running the simulation framework. The time occurrence of the scenario needs to be specified as well. Not only values, Kaala 2.0 also support videos matching the scenario.

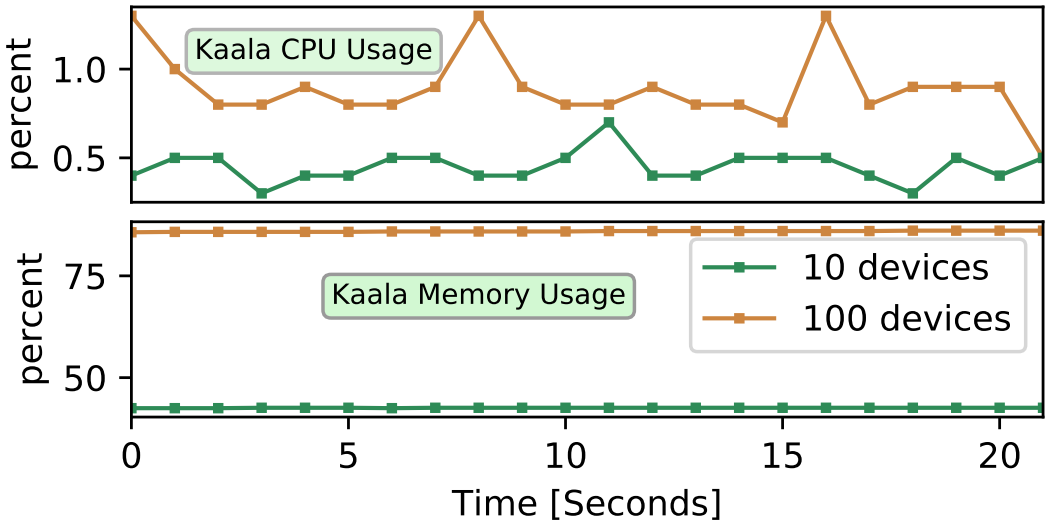


Figure 4.6: Kaala 2.0 Performance Evaluation - Docker

Kaala 2.0 support simulation of IP cameras and a default video will be played when the IoT device is started. During the fire event, a fire video can be specified in the configuration file and the simulated IP camera will start streaming the video with fire. By this design, various scenarios can be simulated in Kaala 2.0. Assume that the quality of video need to be changed based on the network bandwidth available or based on time. This scenario can be easily simulated in Kaala 2.0 using the scenario configuration.

4.4.3 High-Bandwidth Data Simulation

Next, in-order to simulate high-bandwidth scenario, we design our simulated IP camera using a Real-Time Streaming Protocol (RTSP) server [54]. The server will listen in a port and the client will be listening in a different port. The producer of the video will connect to the server port and the consumer of the video will be connecting to the client port to play the video. Both the producer and consumer can be designed to run in any host, so that the traffic flows through the network. More about RTSP implementation and evaluation is discussed in §4.5 and §4.6.3.

4.4.4 NextG Network Support

In order to support NextG simulation, Kaala 2.0 is designed to support regular network connection to the cloud as well as connecting to the cloud through RAN and 5G Core. In case of NextG, each IoT device will connect to the 5G network as an UE. The design supports connecting the IoT gateway to the RAN as a UE or each IoT device as an UE to the RAN. If each IoT device acts as an UE, then those devices wont be able to integrate with a IoT gateway, it needs to connect to the IoT cloud through the 5G Core. Once the UE is connected to the 5G Core, then it can start sending the IoT related data to the IoT cloud through the RAN and 5G Core. Each IoT device will have an option to be a regular IoT device or NextG capable device. Kaala 2.0 supports both simulated or real-world UE's. The real-world UE's can connect only to real-world RAN and the simulated UE's can connect only to simulated RAN due to the radio conditions. Because in simulations, the radio conditions are simulated as well. The real-world RAN can connect to both simulated or real-world 5G core based on the system setup.

4.5 Implementation

We used mininet [52] and docker framework [53] to simulate the IoT devices. The simulator framework can simulate both the generic IoT devices and the vendor-specific IoT device. Since we are leveraging the mininet framework, each IoT device runs in its own process and gets dedicated network characteristics. We used the NodeJS version of the mininet framework to simulate the IoT devices. And for the docker version, each IoT device runs in its own container. The generic IoT device uses the basic MQTT client and the vendor-specific IoT device uses the respective vendor specific client SDK to communicate to the IoT gateway. The IoT gateway information including the authentication details required to connect to the vendor-specific IoT gateway are passed as parameters while starting respective vendor-specific simulated IoT devices, so that each device knows which IoT gateway they need to connect, authenticate and communicate with. Each simulated IoT device also consists of a profile which tells the type of device it simulates and the list of properties associated with the IoT device. Using this implementation, we were able to simulate vendor-specific IoT devices and at the same time, provide dedicated network resources to each of the IoT devices.

The simulation framework loads a configuration file during startup. This configuration files includes the list of IoT devices which needs to be simulated. And the support for new IoT devices can be easily added to Kaala 2.0 by just adding a profile for the newly added IoT device. The new IoT device can either use the generic application logic which sends data periodically based on the configuration which is discussed next or implement its own application logic. Each IoT device entry in that list contains a list of properties and these properties can be easily extended for new properties. Some of the key properties include the name of the IoT device, the profile (light, HVAC, smoke sensor, etc.) of the IoT device, the type of device (generic or vendor-specific) and finally the list of properties and the respective time-interval to report to the IoT gateway. If the device type is vendor-specific, then the authentication information including the certificates are passed as arguments through the proposed design.

There is a another configuration file which is called as scenario configuration file. The main purpose of this configuration file is to configure when the scenario need to be executed, the list of IoT devices properties which need to be included in the scenario

and the values of the properties of those IoT devices during the scenario. The default values need to be specified at the end of the scenario based configuration file in-order to complete the scenario. For example after the fire in the room scenario finished running, either the next scenario needs to follow in the configuration file or the normal values added for the simulated IoT devices to continue execution. Otherwise, the simulation framework will keep simulating the values of the fire in the room scenario.

As discussed in §4.4.2, there are two main components in simulating an IP camera, the producer and the consumer. The video which need to be streamed in the IP camera need to be configured in the simulation configuration file along with the time of stream. Based on the configuration, Kaala 2.0 will connect to the RTSP server to send video data. The producer keeps producing the video to the RTSP server by connecting to the local RTSP server port and the consumers can consume the video by connecting to the client port of the RTSP server of the respective IP camera IoT devices.

The main advantage of Kaala 2.0 over IoTNetSim is that, IoTNetSim creates virtual images for each IoT device but Kaala 2.0 uses processes for each IoT device. We tried to perform additional performance evaluation with IoTNetSim but due to the design of IoTNetSim, we were not able to control the periodic interval of the simulated sensors within the IoTNetSim framework. IoTNetSim does not simulate close-to-real devices. The definition of close-to-real devices is that, when connected to an IoT gateway or IoT cloud, the IoT gateway or IoT cloud will not be able to differentiate whether the connected client is a real or simulated IoT device.

We leveraged Open Air Interface's (OAI) [55] UE, RAN and 5G Core modules for Kaala 2.0. Each of the IoT device will run OAI's UE module. In order to simulate 5G networks, the 5G Core and RAN are started in sequence. Then the IoT devices are deployed and the UE module in the IoT device connects to the configured RAN accordingly. If a IoT device is configured to be a NextG capable, then it will acts as a UE and try to connect to the 5G RAN else it will try to connect to the cloud using regular IP network.

4.6 Evaluations

In this section, we seek to understand the scalability and performance of Kaala 2.0. We connect simulated IoT devices to a real networks (we use Amazon AWS as case study) and finally evaluate a scenario based simulation.

4.6.1 Scalability and Performance

Since the simulation are usually run in a machine by the developer or tester, we want to understand the scalability and performance of running Kaala 2.0 in a regular machine. For the experimental setup, we used a Linux Virtual Machine (VM) which was allocated 4GB memory, 2 processors and 20GB for storage. The VM was running Ubuntu. To understand the memory consumption and scalability capability of Kaala 2.0, we conducted two experiments; In the first and second experiment, we simulated 10 and 100 IoT devices respectively. The simulated IoT devices were a combination of temperature sensor, smoke sensor, humidity sensor, flame sensor and motion sensor. Each of these sensors publish their data based on their IoT device profile every 15 seconds. The data in Fig.4.5 shows that as the number of sensors increases, the processor and memory usage increases significantly. Kaala 2.0 is not just a simulator, it is an emulator as well. Each sensor runs its application logic in an individual process. So the processor and memory usage are expected to increase as the number of devices scale up. This is because as the number of IoT devices increases, more processes are created.

Since Kaala 2.0 uses mininet and docker framework for simulating IoT devices, for each IoT device, mininet and docker framework creates a network interface. So as the number of IoT devices increases, there is time taken to configure a new network interface in the host machine, get a new IP address and bring the interface up. The initial spike in both the memory and processor usage shows that both the mininet and docker framework consumes both memory and processor to create and setup the virtual network. And an additional reason for the spike is due to the spawn of processes or containers for each of the simulated IoT devices.

Table 4.1: Comparison of Kaala 2.0 with IoTNetSim

Features	IoTNetSim	Kaala 2.0
Vendor-Specific IoT Devices	✗	✓
MQTT Protocol Broker	✗	✓
Cloud Layer	✓	✗
Semi-Real IoT Devices	✗	✓
5G Capable Scenarios	✗	✓
5G RAN	✗	✓
5G Core	✗	✓

4.6.2 Interacting with Real Systems using 5G Networks

For this experiment setup, first we created a user profile in the AWS IoT [56]. Then configured an IoT device along with the necessary security certificates which are required to authenticate and connect to the AWS IoT Cloud. The security certificates are downloaded in the host machine in which Kaala 2.0 is running. Next, in Kaala 2.0 configuration, we specify that a vendor-specific IoT device need to be simulated and the path to the downloaded security certificates are configured as well. The IoT device can be of any profile because this is a proof-of-concept experiment scenario. These steps can be repeated for any number of vendor-specific devices. The same steps can also be followed for different vendors like Microsoft Azure or Google IoT clouds as well. Then, we start the simulation framework. Based on the loaded configuration, Kaala 2.0 knows that a vendor-specific IoT device needs to be simulated and the application in the simulated devices will try to connect to the IoT gateway or IoT cloud using the provided security contexts. Once connected, the simulated IoT device will start publishing the data. Particularly, in this experiment, all the data is published to AWS IoT cloud.

4.6.3 Scenario-based Event Simulation

In this section, we assess the scenario-based event simulation in Kaala 2.0. First, for fire in the room scenario, the necessary IoT devices were configured via the Kaala 2.0 configuration file. This configuration file is also a pre-configured profile in Kaala 2.0. Initially, all the IoT devices will be publishing respective data to the IoT gateway or

IoT cloud and the data will be related to normal operation in a room. In the scenario configuration file, the time to start the scenario based simulation will be specified. At that time, each IoT device property configured in the scenario configuration file will be configured with the value specified in the scenario configuration file. And the values gets changed synchronously across all these IoT devices. The flame sensor will report 'true' stating that a flame is detected. The temperature sensor shows a significant increase in the current temperature. And also a video in which fire is shown is played exactly at the same time. Additionally, there can be two sub-scenarios on fire. First one is to make the motion sensor detecting a movement in the room and the next one with motion sensor not sensing any movement or person in the room. This will help to validate scenarios like what happens when a person is in the room during the fire event and what happens when there is no person inside the room when the fire event occurs.

4.6.4 High-bandwidth Data Generation

Next, sending high-bandwidth data was validated. As discussed in §4.4.2, Kaala 2.0 have an RTSP server running when simulating an IP camera IoT device. The server and client port of the RTSP server are configurable in the IP camera IoT device profile. As of now the simulated IP camera IoT device can run server and client on the same port number. This can be a potential future work to support different ports for different IP camera IoT devices. A 8K video is sent to the simulated IP camera by connecting to the server port of the RTSP server. And the consumer consumes the video by connecting to the client port of the RTSP server which is running in the simulated IP camera IoT device. Both the producer and consumer were ran in a different host other than the IP camera IoT device, so that the traffic is flowed in the network. The video being played by the producer can be completely controller by the application using the simulator configuration file. And also the timing of different scenarios can be controlled and configured by the scenario based configuration file. We also simulated a scenario in which the quality of video changes based on the time. First 30 minutes, the producer was configured to produce 8K video and then for the next 30 minutes a 4K video was produced. This proves that Kaala 2.0 is capable of simulating videos of different qualities to test different scenarios of streaming applications.

4.6.5 NextG Simulation

During the startup, an instance of 5G Core and RAN are started and the RAN is connected to the 5G Core. In the configuration file, if the IoT device is configured as NextG capable, then the IoT device acts as a UE and successfully connected to the configured RAN. Once when the UE is successfully attached to the RAN, the IoT data is sent through the RAN and 5G Core to the IoT cloud. Support for multiple UE's connecting to the 5G core was evaluated and the data flow from the UE to the IoT cloud through the RAN and 5G Core was evaluated as well.

4.7 Related Work and State-of-Art

In this section, we discuss related works and especially discuss the Start-of-art simulator we used to compare Kaala 2.0 with in § 4.6. IoT simulators can be grouped into the application layer, Network layer and Cloud layer simulators. Each of these simulators serve different purposes and can be used in different IoT solutions. We discuss a few simulators per layers hereon.

Cloud simulators: In a recent survey [49], the authors identify IOTSim [57] as a Cloud layer simulator widely used in IoT research today. IOTSim focuses on replicating the diversity and heterogeneity that exists amongst IoT devices while generating big data for processing using MapReduce model in the cloud. IOTSim is based on CloudSim [58], another popular IoT cloud simulator platform. The key design goal of CloudSim is to model service brokers, different cloud policies within an IoT cloud infrastructures. IOTSim extends CloudSim with IoT application modelling support and enables big data processing in cloud computer environments. Other Cloud based simulators in the literature are GreenCloud [59] and iCanCloud [60]. GreenCloud is an energy-aware cloud simulators for packet-level research. These cloud simulators are widely used in the IoT research. Nonetheless, they all lack IoT network connectivity and the IoT network layer modelling.

Network simulators: Like NS-3 [61], OMNeT++ [62], iCanCloud [60] have been widely used in wireless sensor networks. iCanCloud focuses on simulation of Amazon instances on the cloud. It does this by implementing a fully flexible hypervisor that

lets users manage the brokering policies within Amazon instances on the cloud by connecting various VMs with the TCP/IP stack. OverSim [63] and PlanetSim [64] are based on OMNet++ and Java respectively and both focuses on simulating overlay networks. PlanetSim is easily extensible, easy to used and learn and easily integrated into other frameworks. All the above network simulators do not follow IoT standards, IoT protocols and IoT radio models.

Application simulators: These simulators focus on modelling IoT protocols like MQTT, CoAP and AMQP and resource management in a controlled environment. However, they lack IoT Cloud layer prototyping. Some examples of IoT application-based simulators in the literature include: MDCSim [65], iFogSim [66], and IOTSim [57]. MDCSim is a multi-tier data center simulator that can be used to develop and study IoT application on a multi-tier cloud architecture. MDCSim implements each individual tier in a flexible and fully configurable manner and supports IoT pub/sub data communication model. iFogSim adapts a Sense-Process-Actuate model in which simulated edge devices publish data via an IoT network for applications running on Fog devices to subscribe to. These fog devices then process and translates these data into actions that are forwarded to actuators. iFogSim takes into account IoT network protocols and make studying the resource management policies (*i.e.*, network congestion, latency (timeliness), energy consumption, and operational cost) with an IoT environment very easy.

Most recently, Maria et al. proposed IoTNetSim [67] to model an end-to-end IoT services in a multi-layered manner. It is a platform that models IoT heterogeneous nodes with all their characteristics like mobility, energy, and profile. IoTNetSim aims at modelling an IoT service across all layers. *i.e.*, From the cloud, Fog, Edge, IoT and application layer with various IoT network designs. It models IoT connectivity using Virtual Machines (VM) with an event-based engine that is configurable. Support for IoT node mobility is implemented via latitude and longitudes coordinates that are read from a .csv file which follows a previous pre-recorded behavior pattern. IoTNetSim uses CloudSim for cloud simulation functions like modularization, large data process and resource management simulators. IoTNetSim sensor data is simulated in real-time and days worth data can be simulated in few seconds.

To the best of our knowledge, a major issue with all current IoT simulators is

their inability to connect to real systems (*i.e.*, part simulation and part real system prototyping interactions). Additionally, they do not support vendor specific IoT (edge) device modelling capable of communicating with vendor IoT gateway SDKs. 5G capable scenarios modelling are not also supported, therefore this chapter sets to address these drawbacks within Kaala 2.0 [68].

4.8 Summary

We have presented *Kaala 2.0* – a modelling, simulation and emulation platform that are capable of creating IoT devices of various types. Using our proposed simulation framework, we were able to simulate multi-vendor specific IoT devices in a single simulation framework. We also simulated real-time events like fire in a room/building scenario and evaluated how this work can be extended for other real-time scenarios. We were able to simulate devices which can generate large amount of data to verify and validate 5G technology.

Chapter 5

HyperRAN: Towards a Fine-Grained, Semantics-Aware, Intelligent NextG Radio Access Network Architecture

5.1 Introduction

Promises of 5G inspired a slew of new applications and services with disparate bandwidth, latency, and reliability requirements. These include volumetric video streaming, Digital Twins, cooperative autonomous driving, and other V2X (vehicle-to-everything) use cases. To meet these requirements, 5G improved upon 4G LTE (Long-Term Evolution) with new capabilities such as intra- and inter-frequency Carrier Aggregation (CA) [69], a "flow-based" QoS (quality-of-service) architecture and network slicing. In addition to low- and mid-band frequency ranges, 5G New Radio (NR) expands into the mmWave high-band frequency range. Future 6G is considering sub-THz and THz frequency bands. While these bands offer far higher data rates, they suffer many issues, such as limited coverage ranges, requiring line-of-sight (LoS), and sensitivity to blockage and environmental factors. Supporting diverse applications with disparate service needs requires not only expanding network capacity, but also *intelligence* and *agility* to

efficiently utilize the scarce radio resources.

We posit that not all data is of equal *utility* to an application. This is true for (volumetric) video, LiDAR, and other sensor data used in augmented/virtual reality (AR/VR), Digital Twins, and V2X applications which require high bandwidth for data delivery. For example, for tele-operations of an autonomous vehicle [70], LiDAR data points within the field of view are in general more critical than, say, those on the right side of the vehicle. Likewise, data points containing moving objects that are potentially *safety-critical* are more important than those containing stationary or background objects (which, e.g., can be accurately predicted via generative AI. Hence in order to effectively utilize the scarce radio resources, we must exploit *application semantics* and endow the RAN with the agility and intelligence, for example, enabling it to intelligently match diverse radio channels with fast varying conditions to application data with differing utilities, and make smart, dynamic radio resource allocations accordingly. This leads us to re-architect the radio network architecture for next-generation (NextG) wireless networks. The **key contributions** are summarized below.

• We layout an overall framework for re-architecting NextG networks (§5.3). It is designed based on four core principles: *application-aware & semantics-guided*, *fine-grained*, *truly cross-layer*, and *programmable*. Our framework enables application endpoint and network collaboration by (i) (adaptively) refactoring, partitioning, and marking application data with *semantic tags* and embedding them end-to-end across the network and down the network protocol stack; and (ii) endowing NextG RAN with the agility to intelligently match available frequency channels with differing characteristics to appropriate data (sub-)streams/objects, and dynamically allocate fast varying radio resources to transport the right (amount/type of) data with the best deliverable utility to an application.

• This chapter focuses on the design of HyperRAN (§5.4), a novel NextG RAN architecture which embodies the design principles outlined above. It is *fine-grained* in that data within the same applications session or flow may be dynamically mapped to different radio channels in accordance with their utility to the application, thus with *differentiated* QoS treatment. This is achieved by embedding application semantics, service, and user contexts across the RAN protocol stack, which enables intelligent

radio resource allocation that takes into account application semantics, service requirements, user context as well as fast varying channel conditions. Hence HyperRAN is both *(application) semantics-aware* and *intelligent*. This is in contrast to the existing 5G "flow-based" QoS architecture where data belonging to the same "QoS" flows are always treated the same inside the RAN such as radio channel and resource allocation (see 2.2). Realizing the proposed HyperRAN architecture has now become feasible by fully embracing and exploiting the "softwarization" and "cloudification" of RAN disaggregation. In particular, our HyperRAN design is O-RAN compliant. By embedding application semantics across the RAN protocol stack with *fine-grained, rule-based control logic*, we program the HyperRAN behavior via O-RAN RICs, and endows it with the agility and intelligence for application-aware, fine-grained, cross-layer decision making.

• At the core of HyperRAN is Hyper Scheduler (§5.4.2) which sits between the Radio Link Control (RLC) and Media Access Control (MAC) (sub)layers of the standard RAN protocol stack (see Fig. 5.1): it can be viewed as part of a new "upper" MAC layer. Configured with one or multiple cells and radio channels (with, e.g., CA), the (low-level) MAC and physical (PHY) layers are responsible for scheduling data transmissions from a set of (*virtual*) MAC queues (or virtual RLC channels) at a faster time scale, e.g., slot- or multiple-slot levels (sub-1 ms (millisecond) or 1ms). In contrast, Hyper Scheduler operates at a slower time scale, e.g., sub-frame or frame-levels (several or 10's ms). It is responsible for dynamically mapping and assigning (user) data to different "radios" (one per each low-level MAC scheduler) based on application semantics, Quality of Experience (QoE), user context, and channel conditions. Hyper Scheduler is *programmable*; its decisions are controlled by *declarative* policies or rules supplied by O-RAN near-real-time RIC that can be dynamically updated, e.g., when the user context such as mobility has changed. In addition, Hyper Scheduler adaptively determines when to re-transmit or discard data. Hence, when radio resources are limited, low-priority or stale data may be dynamically discarded.

• We implement a "proof-of-concept" prototype of HyperRAN with the basic building blocks including Hyper Scheduler using srsRAN [71] as the code-base. We reuse and set up multiple (slightly modified) PHY/MAC layers to emulate a multi-band RAN, and re-architect and modify the RLC/Packet Data Convergence Protocol (PDCP)/Service Data Adaptation Protocol (SDAP) modules.

• For evaluation, we consider two emerging use cases with high-bandwidth and interactive low latency requirements – ultra-high-resolution volumetric video and LiDAR data streaming for co-operative driving. We set up a testbed with HyperRAN and multiple emulated UEs (user equipment) and conduct experiments under various settings using real-world radio channel traces collected from commercial 5G networks. Our experimental evaluations show HyperRAN provides significant improvements to volumetric video streaming and LiDAR streaming: For volumetric video streaming across stationary and mobility scenarios, HyperRAN significantly outperforms existing solutions such as end-to-end bit rate (End-point) adaptation, CA and Static Channel Mapping by significant margins. For example, it reduces video stall time between 46-75% under stationary and 22-64% for mobility cases through improved buffer occupancy for critical base layers. For LiDAR streaming, HyperRAN's flexible and contextual prioritization and discard policies netted a 50x reduction in LiDAR packet delivery time, a 7.7% reduction in overall LiDAR packet loss compared to baseline results. Further, under challenging scenarios with multiple UEs running volumetric video streaming and LiDAR streaming, HyperRAN leverages application-aware traffic prioritization for intelligent decision making, providing roughly 15% improved delivery rate with 2-4× reduction in latency.

In a nutshell, we aim to fully exploit the benefits afforded by "softwarization" and "cloudification" to re-architect NextG RAN to meet future application demands. The proposed HyperRAN is a first step towards this goal, as there are many challenges yet to be solved. We envisage HyperRAN to be first deployed in "private" NextG networks for new (vertical) industrial use cases (cf. §5.3.5).

5.2 Case for HyperRAN: Why Existing Solutions are Inadequate

We present two use cases to argue why existing solutions are inadequate to support future applications, which motivate our proposed HyperRAN architecture. In §5.6 we will employ these use cases to demonstrate the limitations of existing solutions and benefits of HyperRAN using "real-world" data.

Use Cases. Consider (a) *cooperative driving* where an edge cloud service streams LiDAR data representing the dynamic 3D environment mapping to assist an autonomous shuttle to "see" objects (e.g., a bicyclist attempting to cross an intersection from another road) that might be obstructed from its field of view. Such a dynamic 3D environment may be constructed from data collected from multiple sensors mounted on the road infrastructure and/or from other autonomous vehicles (AVs). Cooperative driving is one of the key use cases that 5G is envisaged to enable. Consider further that (b) Alice, a passenger, is streaming a volumetric video, e.g., as part of an extended reality (XR) or metaverse application while riding in the shuttle. The 5G radio cell towers (gNBs) along the shuttle route are often configured with multiple cells (and radio bands). However, as the shuttle moves along the roadway, the number of cells that it is under coverage may vary, and more importantly, the radio channel conditions can vary significantly. As a result, the total bandwidth demand of each use case individually (or together) may frequently exceed the radio resources available [72, 73]. What can we do to ensure good quality-of-experience (QoE) for such use cases?

Why End-to-End Adaptation Inadequate? The classic approach to tackle this fundamental *demand-resource mismatch* problem is to rely on end systems to perform bit rate adaptation. For example, current video streaming services employ an adaptive bit rate (ABR) algorithm that adapts to changing network conditions by selecting video chunks encoded at different bit rates based on measured network throughput. For use case (a), we may divide LIDAR data into sectors per "frame" (360° scan) and deliver only a subset of the sectors based on the *estimated* available bandwidth at the end system (e.g., the edge cloud service). For volumetric video streaming in use case (b), it is perhaps more natural to employ scalable video coding (SVC) (see, e.g., [74, 75, 76]) to progressively encode 3D video frames using multiple layers. As first pointed out in [72], relying on *end-to-end* bit rate adaptation may not be effective due to several reasons. First, due to wildly varying channel conditions and fluctuating bandwidth (especially when a user is mobile, e.g., driving), it is difficult to measure and estimate the network throughput accurately for bit rate adaptation. Second, and perhaps more importantly, the channel conditions vary far faster than the end-to-end bit rate adaptation cycles. This may lead to two bad effects: (i) if the end system decides the network bandwidth is sufficiently high to accommodate more LiDAR sectors or video layers, the channel

condition may have become poor when the data arrives at the 5G RAN. (ii) If the end system decides the network bandwidth is low and it thus selects fewer sectors or layers to transmit. However, a previously moving object that blocked the light-of-sight (LoS) path of, say, a mmWave channel, has moved out of the way, and thus the channel condition becomes good again. This leads to a loss of opportunity to transmit more data. In §5.6, we conduct experiments that indeed confirm the inadequacy of end-to-end available bit rate adaptations under fast varying channel conditions.

Why the Existing 5G QoS Framework Also Inadequate? Using the 5G QoS framework, we can, for example, transmit each LiDAR sector or video layer as separate "QoS" flows and mark them with different QFIs (thus associating them with different 5QI profiles/QoS treatments). For example, we can assign QoS flows containing LiDAR sectors in the front view of the shuttle with higher priority and better QoS treatments than those containing other sectors. Likewise, we can assign the QoS flow containing the video base layer with the highest priority and QoS class, while other layers with decreasing priorities and QoS classes. When the available radio resources are insufficient [77] to meet the bandwidth demands of all the QoS flows, *in theory* the 5G RAN can deliver only higher priority flows, dropping data from the lower priority QoS flows. However, *in practice* several major issues arise. First, QoS flows are mapped to fixed DRBs with pre-configured QoS parameters, which are then mapped to (fixed) RLC and logic channels (MAC queues), and finally transport and physical (radio) channels. If one radio channel always outperforms other channels, this would not have been an issue. Unfortunately, this is not the case in general (see Fig. 5.10 for real channel quality indicators (CQI) measurement results). It is in general difficult, if not feasible, to dynamically associate different radio channels to DRBs (thus QoS flows), based on, e.g., channel conditions. One might be able to reconfigure the QoS-flow-to-DRB or DRB-to-channel mappings via RRC (semi-dynamically); Availability of such function is often *vendor-specific*, and can only be performed in an *ad hoc* manner. Further, such re-configuration or channel re-mapping cannot be done in an *application-specific* or *context-dependent* manner. For example, when the shuttle is at a bus stop with LoS to a gNB, we may prefer to use the available mmWave channel to "burst" a large amount of higher-priority lower-layer video data [72] whereas when the shuttle is driving, we may prefer to use mid-band 5G channels for lower-layer video data. Whereas for cooperative driving, we may also

prefer to use 5G low-band channels for safety-critical V2X data due to their better reliability and large coverage, while always mapping LiDAR data to mid-band channels in accordance with the data priority and channel conditions. Moreover, the 5G flow-based framework may still be too *coarse-grained* for many emerging applications. For example, for LiDAR data, even after dividing into sectors, the bandwidth requirement of a single sector (QoS flow) may still be too large to be met. However, not all data points in the sector may be of significance – when available radio resources are not sufficient to meet all bandwidth demands, ideally only data points that cover objects of interest need to be delivered. This cannot be done in today's 5G RAN. All in all, we need general *programmable* interfaces to control the RAN "internal" behavior dynamically, in accordance with application semantics, radio channel conditions and user mobility, environmental and other contexts (e.g., the shuttle is stationary at a bus stop).

In summary, we argue that existing 5G networks are too rigid to provide the agility and intelligence to make dynamic decisions to adapt to fast varying radio channel conditions in accordance with application semantics, service requirements and user/environment contexts. More specifically, the existing 5G RAN architecture suffers from the following limitations. **1) Inflexible, Implicit Static Mapping.** As discussed in §2.2, the existing 5G RAN architecture, QoS flows are mapped to DRBs with pre-configured QoS parameters. DRBs are bound to the lower-(sub)layers channels. In other words, the QoS treatments of DRBs, e.g., bit rate guarantees, are *implicitly* passed down to the MAC layer (e.g., via "hard-coded" scheduling weights) for radio resource scheduling. **2) Stale Data.** As an example of such inflexibility, the MAC scheduler is obliged to transmit "stale" packets that have been buffered despite they have passed their "useful" deadlines. This not only wastes scarce radio resources, but also further impedes the timely delivery of future (still "useful") data. **3) Not Fine-Grained.** The flow-based QoS framework in 5G treats all data within a flow, e.g., a video stream, the same way, despite that there are I, P, B frames or base and enhanced layers of different utilities to the application QoE. **4) Not Programmable.** All in all, the RAN functionality and features are hard-coded and not programmable. **5) Vendor-specific, Closed Implementation Limiting the Potential of O-RAN.** Despite the fact that O-RAN has defined *open interfaces* (e.g., the E2 interface) for "intelligent" RAN control, without an *open & programmable architecture*, RAN implementations are largely vendor-specific

and closed. This severely limits the potential of O-RAN (see also footnote 3 in Chapter 2). We believe that the research community can play a vital role in further advancing 5G to NextG by fully leveraging the potential afforded by the *disaggregated* and *softwarized* RAN architecture.

5.2.1 HyperRAN Deployment Challenges and Opportunities

As a new RAN architecture design, HyperRAN will likely encounter many practical and engineering challenges in future deployment. For instance, diverse business agreements may be necessary to support different services from each operator. HyperRAN might also need to address the requirements of numerous active services that prefer dedicated channels. It's important to note that these challenges are orthogonal to the core technical challenges we aim to tackle and introduce. Moreover, they are not new or unique to HyperRAN; for instance, the introduction of network slicing has not deterred 3GPP or mobile operators from embracing the concept. The ongoing trends of "softwarization" and "cloudification" in O-RAN have significantly reduced hurdles to system integration and deployability. As such, we argue that HyperRAN can be incrementally deployed, and *scaled by running more instances*. For example, HyperRAN can be deployed in parallel to a "conventional" 5G RAN to support a subset of applications/services for which the cellular provider has established service agreements. In private NextG networks, HyperRAN can be deployed to support industrial "vertical services" such as tele-medicine [78], smart warehouse [79, 80], and robotaxi [70].

5.3 Framework Overview

The HyperRAN compromises UE, RAN, O-RAN RIC, 5G Core, external data network, application client, and application server. Some components were modified and some of the components are introduced as part of HyperRAN. The modified components which were done following the architecture and design of existing components. In the next sub-sections, we describe the design change needed for each of the identified components.

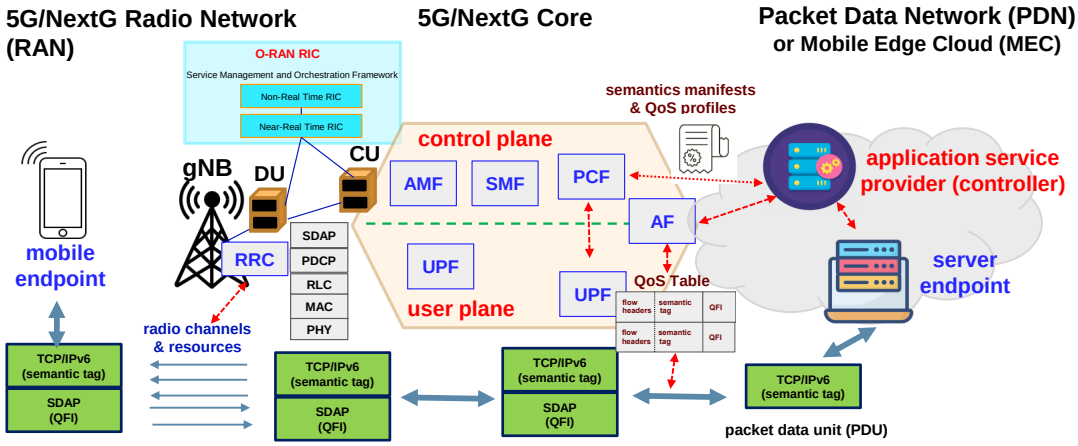


Figure 5.1: Semantics-Aware, Fine-Grained, Cross-Layer, Software-Defined NextG Framework.

5.3.1 Design Principles and Architecture

We lay out four core principles that guide the design of our proposed NextG framework below:

• **Application-Aware & Semantics-Guided.** With growing demands for bandwidth, we believe that *intelligent* radio resource allocation and scheduling must be *application-aware*. As not all data is of equal utility to an application, decision-making must be guided by *(application) semantics*. This requires cooperation from application service endpoints.

• **Fine-Grained.** Unlike the existing 5G QoS architecture which is flow-based (see §2.1), our approach is more *fine-grained*. Application endpoints can (dynamically) partition data into smaller data chunks, objects or sub-streams ("subflows"), and mark them with appropriate *(service-specific) semantic tags*. At the finest granularity, our approach allows per-packet (differentiated) QoS treatment. Such a fine-grained approach allows us to more efficiently utilize scarce radio resources.

• **(Truly) Cross-Layer.** Application semantics information is not only passed down from the application to the IP layer, but more importantly, also embedded across the

NextG HyperRAN (sub-layer) protocol stack. Such information will be used by our novel *Hyper Scheduler* running at the (upper) MAC sub-layer to intelligently match application data (of various utilities) to diverse radio channels and enable (radio/cell-specific) MAC schedulers to allocate radio resources accordingly.

• **Software-Defined.** Our framework is *software-defined* and *programmable* in that it follows the same principles of software-defined networking (SDN), where the behavior of data/user plane functions is controlled and programmed by the control plane. However, our design is more flexible and fine-grained. In NextG core networks, we mark data packets via *service-specific QoS tables*, extending the SDN flow tables. In NextG (Hyper)RAN, we program the Hyper scheduler via (service-specific) policies or rules supplied by O-RAN RICs.

The overall architecture of our framework is schematically sketched in Fig. 5.1, where we have depicted the relations between a NextG carrier (with its constituent core network, RAN, and O-RAN controllers) and an application service provider (with its service controller and server/client endpoints). Next, we briefly describe the major functions of each key component.

5.3.2 Application Service Endpoint Functions

Our basic premise is that an application service provider (ASP) enters into a *cooperative service level agreement* (SLA) certain financial or other arrangements) with a NextG carrier to *collaboratively* provide support for its application or service over the NextG network. An *ASP controller*, e.g., residing within a mobile edge cloud (MEC) close to the NextG network, supplies the NextG network orchestrator with (*service-specific*) application semantics manifests and QoS profiles, e.g., specified similar to the service (abstraction) models defined in O-RAN SMO or ONAP [81]. The ASP may also supply the NextG carrier with *application functions* (AFs) for service-specific data processing, e.g., data classification, filtering and mapping of semantic tags to (service-specific) QFIs. We note that all these operators are performed either at the time of service creation or at the time of user (PDU) session establishment, with appropriate rules/policies pushed to the relevant network elements.

At the time of data generation and delivery, the ASP application *endpoints* will mark (*fine-grained*) data objects or sub-streams (video layer frames/chunks or LiDAR

sectors/objects) with appropriate semantic tags for the (desired) QoS treatments over the NextG network, in accordance with a pre-defined application (semantics) schema. We note that such tasks can now be performed readily and automatically using AI algorithms, e.g., for object recognition. Application semantic tags can be embedded in the transport or IP layer headers in multiple ways[1]. In Fig. 5.1 we assume that new P4 headers are defined to carry semantics tags fields. In our current prototype implementation for evaluation, we use an IPv6 extension header to encode application semantics tags.

5.3.3 NextG Core and RAN Networks

Based on the SLA with the ASP, the NextG network may implement the application service in a (dedicated) *network slice* or treat the service "normally" without allocating a network slice. In either case, the NextG core network will institute appropriate control functions, such as SMFs and AMFs to set up and authenticate packet data network (PDN) sessions for the application flows and track the mobility of mobile client endpoints. In particular, the Policy Control Function (PCF) will define policies and rules for classifying and mapping (ASP-specific) application semantic tags to (service-specific) QFI values. These rules are implemented as *QoS tables* (extending the SDN flow tables). Each entry of a QoS table is of the form $\langle$ flow header; semantic tags | QFI $\rangle$. In other words, besides the "standard" headers used in SDN for flow matching, the QoS table also includes (service-specific) semantics tags for data classification and mapping.

Upon receiving the data from the application, the NextG user plan functions (UPFs), possibly assisted by service-specific AFs, will classify the packet data units (PDUs) using the QoS tables and encapsulate them in the SDAP headers with appropriate QFI values. Once reaching the NextG radio access network, HyperRAN will map the QFI values to metadata associated with PDUs in accordance with the ASP service context (configured via O-RAN). The metadata will be embedded and passed down the HyperRAN protocol

[1] For example, we can embed the semantics tags in a new QUIC extension header and use QUIC as the transport layer protocol. We can also embed them using the DSCP (Differentiated Service Code Point) bits in the IPv4 header. We can also re-purpose the IPv6 flow label or use an IPv6 extension header to carry the semantics tags. More generally, we can use the P4 language to define new headers with new semantics tags fields as well as to specify (application-specific) rules and write programs for integrated application/transport/network layer data packet processing.

sub-layers and utilized by the Hyper Scheduler for intelligent radio channel mapping and radio resource scheduling (see §5.4 and §5.4.2 for more details).

5.3.4 O-RAN SMO, Non-RT and NRT RICs

Our design follows the specifications and guidelines from O-RAN. For example, non-RT RIC is responsible for configuring the Hyper Scheduler. The nrt-RT RIC will provide policies and rules, and periodically update them (e.g., using an AI algorithm), to instruct and influence the behavior of Hyper Scheduler. The role of O-RAN will be discussed further in §5.4 and §5.4.2. There are many challenges and issues yet to addressed to realize the proposed framework *end-to-end*, e.g., design of application schema for (dynamic) data partition and application semantics tagging (cf. §5.3.2 and §5.6), due to space limitation we will not elaborate here. The remainder of the chapter focuses on the challenges in incorporating *semantics-aware, fined-grained intelligent control* in radio access networks.

5.3.5 Targeted Use Cases and Deployment Scenarios

HyperRAN is especially designed to enable emerging/future use cases such as XR/ multiverse, cooperative autonomous driving, Digital Twins that require not only ultra-high bandwidths (from 100s Mbps to several Gbps of "raw" data throughput) but also low latency (at the time scales of 10s to 100s ms[2]). Furthermore, for use cases with more stringent latency requirements, HyperRAN can readily incorporate 5G URLLC *as it*, if supported by the underlying MAC/PHY sublayers, since HyperRAN utilizes existing MAC/PHY for its lower layer functions (§5.4.2). It can further provide *added benefits* of traffic prioritization for URLLC communications based on application needs. As stated earlier, we expect HyperRAN to be first deployed in "private" NextG networks (or as separate RAN instances running in parallel to existing 5G RAN instances) to support

[2]The latency requirements are congruent to dynamics in typical *machine-environment interactions* and *human-machine-environment interactions*, where objects typically do not move at the sub-ms speed, and therefore radio channels do not vary faster than such time scales and conform to the human perceptual/cognitive needs and interactive control requirements. In contrast to the 5G URLLC (Ultra-Reliable Low-Latency Communication) service which is designed for low-bit-rate, sub-ms (machine-to-machine) communications, supporting these new uses cases with both high bandwidth and low latency requirements is more challenging, due to scarce radio resources and more dynamic (radio) environments they operate in.

emerging vertical industrial use cases. We elaborated on these points in §5.2.1.

5.4 HyperRAN Architecture and Hyper Scheduler Design

We provide an overview of the proposed HyperRAN architecture, highlighting the key innovations we advance to support our design goals (cf. §5.3.1). We then delve into the design of Hyper Scheduler.

5.4.1 HyperRAN Architecture and Innovations

We adopt O-RAN's disaggregated RAN specification and split the main RAN functions into O-CU (central unit) and O-DU (distributed unit). To address the challenges highlighted above and fully take advantage of the software nature and *multi-core* servers on which CUs/DUs will be hosted, we completely re-architect the NextG RAN architecture to enable flexibility and programmability while also enhancing efficiency and scalability. To this end, we introduce several key innovations. While some of the ideas and mechanisms have been widely adopted in other settings (e.g., cloud computing and 5G core), our novelties lie in applying and extending them creatively to tackle the unique challenges in RAN designs.

First of all, we explicitly separate *data* (which holds user PDU sessions/flows), *compute* (which executes various protocol processing entities or functions) and *state* (which holds *control* and other *metadata* and governs the behavior of protocol functions) [3]. By consolidating user data in a *shared packet ring buffer* and employing Data Plane Development Kit (DPDK) as well as other SmartNIC functions available in modern multi-core servers, we develop an efficient software packet processing pipeline with *kernel bypass* and *zero copying*. Associated with each user PDU is a *configurable* metadata container that can be used to pass along information across the layers, e.g., flow tags that can facilitate fast table look-up, semantic tags for (adaptive) QoS treatments, timestamps

[3]This is in contrast to existing open-source 5G RAN reference implementations [71, 55], which, as a vestige of existing 5G and previous generations' *hardware-based* RAN architectural implementations, organize data processing around the RAN protocol layer processing "entities" (e.g., PDCP, RLC entities) as specified in 3GPP specs. Such design couples user data with the compute processes ("protocol processing entities"), producing many inefficiencies and making it harder to scale.

for dynamic packet scheduling to meet latency requirements or packet dropping when deadline has been exceeded. We introduce a set of (hierarchical) index queues containing (packet header) references to logically group and track individual user PDU sessions/flows (i.e., they form "virtual queues"). This is illustrated in Fig. 5.8. As in most cloud computing systems today, we organize compute resources into a pool of (preconfigured) worker threads associated with the CPU cores, further treat and decompose protocol processing entities as a (configurable) sequence/graph of *modular* protocol functions or (compute) tasks, and dynamically schedule and map the modular protocol functions (tasks) to the worker threads/CPU cores. This not only eliminates the overheads of process context switching, spawning or killing CPU processes, but also makes it easier to scale the protocol processing. For instance, if one PDCP entity becomes overloaded, more worker threads/cores may be allocated to process its constituent QoS flows. We explicitly decouple the *state* from protocol functions so that the state can be separately managed. This enables resiliency and scalability (e.g., by appropriately replicating the state to scale out). Perhaps more importantly, it makes the behavior of protocol functions *programmable*, as we will further illustrate below. Last but not least, we introduce a novel Hyper Scheduler in the MAC layer that sits on top of (traditional, now "low-level") MAC schedulers (see below and §5.4.1 for more discussion). The HyperRAN architecture is schematically depicted in Fig. 5.2. We briefly summarize the key O-RU and O-DU designs below.

HyperRAN O-CU. The packet ring buffer contains both DL and UL data from/to 5G UPFs and to/from O-DUs, both communicated via GTP-U tunnels. *Logically* there is a RX (receive) ring buffer and a TX (transmit) ring buffer. On the receiving side, the DPDK-based packet processing pipeline receives packets from UPFs and O-DUs, processing and placing them into the (logical RX) ring buffer. On the transmitting side, it removes packets from the (logical TX) ring buffer, process and transmit them. The core orchestration and task scheduling maps protocol processing entities/functions (tasks) to CPUs and worker threads. For example, packets from user data streams (or "flows") are processed by an SDAP entity which, besides attaching an SDAP header with an appropriate (*service-specific*) QFI value, tags the individual packets with "semantic tags" stored in the metadata associated with the packets. Note that unlike existing 5G QoS flows, packets belonging to the same HyperRAN "flow" or data stream may carry

different (*service-specific*) QFI values. The "semantic tags" may be used to facilitate fast *packet processing rule* (PPR) table lookup for, e.g., subsequent PDCP processing; they may even be passed down to O-DUs for appropriate RLC/Hyper Scheduler processing. By embedding and encoding application semantics via metadata to influence QoS treatments across the protocol stack, our design is thus truly *cross-layer*. The (service-specific) PPR table is part of the state maintained by O-CU. Additional state information includes the service context, user context, PDU session context, and QoS context. They are installed, removed, configured and (semi-dynamically) modified via RRC and O-RAN RICs. By adopting a *software-defined*, *rule-based* paradigm, we make the behavior of protocol functions *programmable*. For example, by installing appropriate PPRs, we can program the PDCP processing entities to whether or not apply header processing to individual packets, what integrity protection and ciphering mechanisms to use, whether or not to re-transmit lost packets, or discard packets that have been buffered for some time, or where to route/re-route packets, and so forth, depending on the service QoS requirements, user context (e.g., user mobility patterns) and application semantics (e.g., how important the data is).

HyperRAN O-DU. The packet ring buffer in the O-DU will only be used for the data to/from with O-CU, as the communications between O-DU and O-RU use the (specialized) open fronthaul interface (7.2 split eCPRI) [22]. Both RLC processing and Hyper Scheduler are *programmable* based on rules configured by RRC and O-RAN (non-RT and nrt) RICs. For example, instead of the three fixed RLC modes – the *transparent* (TM), *unacknowledged* (UM) and *acknowledged*) (AM) modes in the existing RAN, we can configure an RLC entity to process individual packets using mixed UM/AM modes based on the associated "semantics" tags. The introduction of a *Hyper Scheduler* makes (*multi-radio*) MAC scheduling *programmable*, without making significant changes to existing complex (low-level) MAC schedulers that are more intimately tied to the PHY layer. Furthermore, it enables us to perform *intelligent* multi-radio scheduling to match application semantics with fast varying radio channel characteristics based on *software-defined* rules or scheduling logic.

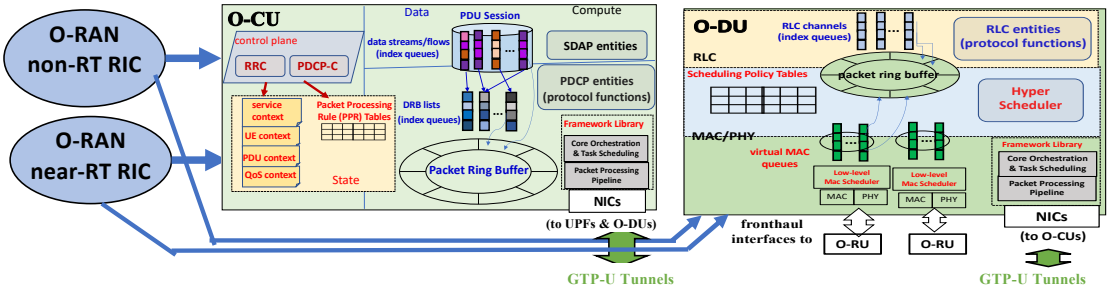


Figure 5.2: HyperRAN Architecture.

5.4.2 Hyper Scheduler Design

In the design of Hyper Scheduler, we assume that there are two or more (*low-level*) MAC schedulers (one for each "radio" or radio technology). The (low-level) MAC schedulers typically operate at much faster time-scales, e.g., at a single slot or multi-slot level (sub-1 ms or 1 ms level). In contrast, Hyper Scheduler operates at a slower time scale, e.g., at the multiple sub-frame (several ms) or frame level (10's ms). As alluded earlier, existing MAC schedulers used in commercial RANs are highly complex and often intimately tied to the PHY layer processing. This is particularly the case when CA of multiple radio channels is employed (within the same "radio"), where time synchronization of different frame structures and other PHY layer issues must be confronted. With Hyper Scheduler, we introduce *intelligence* and *programmability* (For example, in our implementation, we are able to re-use the MAC/PHY implementations of existing open source RAN platforms [55] [71] with slight modifications to the interfaces with the RLC layer). As illustrated in Fig. 5.2, Hyper Scheduler takes data from the RLC channels, map and schedule data from these channels to the low-level MAC schedulers. We maintain a list of (virtual) MAC queues per each (low-level) MAC scheduler. These MAC queues (a list of *virtual* RLC channels) serve as the interface to each low-level MAC scheduler.

Table 5.1: Example Hyper Scheduler Policy Table.

NAME	#[UE+SID]	QFI	UE-Context	CQI	PD
EP1	#[46, 3]	10, 20	Stationary	≥ 10	Prefer Bandwidth
		10, 20	Walking	≥ 5	Prefer CQI
		30, 40	Walking	≥ 0	Drop if Deadline Passes
EP2	#[47, 4]	10, 20	Stationary	≥ 0	Prefer CQI
		10, 20,30,40	Driving	≥ 0	Drop if Deadline Passes

Declarative Policy-Based Decision Engine. Hyper Scheduler applies *service-specific* policies to map and schedule user data to radios/radio channels that account for service requirements, application semantics, radio characteristics (e.g., band and coverage range), dynamic channel conditions and user context (e.g., mobility patterns). The policies are *declaratively* specified (using the *extended* E2 interface) and supplied by the O-RAN non-RT RIC; and they can be dynamically updated by O-RAN nrt- RIC. Hence the behavior of Hyper Scheduler is software-defined and programmable. Using layered (volumetric) video streaming as an application use case, Figs. 5.3 & 5.4 shows two example policies specifications, where there are two radios. In *Example Policy* 1 (EP1), a) when the UE is stationary and the CQIs of both radios ¿ 10, the base layers video streams (indicated with QFI=10,20) always prefers the radio with highest bandwidth; otherwise, it prefers the radio with the highest CQI; the enhancement layer video streams (QFI=30,40) are transmitted using the second radio; b) when the UE is walking, the base layers are always delivered using radio 1 (which has larger coverage) unless its CQI ¡ 5; otherwise, it is assigned to radio with highest CQI is preferred; enhancement layers are scheduled using the second radio. In terms of *prioritization and discard* (PD) handling, base layers video data is never discarded; whereas enhancement layer video data will be dropped if the delivery deadline (20 ms) has passed. For enhancement layer data, the radio resource allocation priority is given in the increasing order of QFI (i.e., data with QFI=30 is first scheduled on the second radio before data with QFI=40, etc.). In *Example Policy* 2 (EP2), when UE is stationary, the base layer always prefers radio with best CQI; if there is not sufficient radio resource, the second radio is also used for the base layer data delivery; whatever remaining available radio resources are assigned for enhancement layer data delivery in the priority order of QFIs. In EP1, the base layer video is only delivered using one radio; the other radio is used for the enhancement layer data delivery. Hence at least two layers of video are delivered simultaneously. In contrast, EP2 may stripe the base layer across two radios to maximize its timely delivery at the expense of no radio resources allocated for the enhancement layer data delivery. These declarative policies are then translated into a *scheduling policy* table (see Table 5.1 for an example) which is used by Hyper Scheduler for periodic radio mapping and scheduling decisions (see below). The PHY layer periodically informs the

Hyper Scheduler of the latest Channel Quality Indicator (CQI) and other channel condition information, e.g., block-level error rates (BLERs); whereas the O-RAN nrt-RIC may dynamically update Hyper Scheduler with the user context when the UE mobility pattern has changed. In §5.6 we will use EP1 and EP2 above (and their variations) to evaluate Hyper Scheduler.

Radio Channel Mapping and Scheduling. Hyper Scheduler performs two basic functions periodically: 1) it *intelligently maps user data to radios* (low-level MACs) using the scheduling policy and logic encoded in the declarative rule-based decision engine; and 2) it schedules (new) user data for low-level MAC transmissions by placing them, dynamically (re-)prioritize packets in the (virtual) low-level MAC queues and discard stale buffered data if necessary (see below). It operates in two stages (pseudocode in Algorithm 1 & 2). In the first *radio mapping* stage, Hyper Scheduler polls the (virtual) RLC channel list and uses the metadata (e.g., flow tags that encode service id, user id and data stream/flow id's as hashes) associated with data packets to look up the scheduling policy table, and use the corresponding decision engine rules to map them to a preferred radio (or a list of preferred radios). Optionally, for each radio, the required or desired bit rates, time sensitivity priority or desired time for transmission, and whether data can be discarded if deadline is past may be calculated and attached as metadata. The output of stage 1 is a (tentative) list of user data radio assignment queues. In the second (*data scheduling*) stage, Hyper Scheduler employs the *weighted proportional sharing* (WPS) algorithm [82, 83, 84] to assign data to radio, taking into account the radio bandwidth available and user data QoS requirements. It first scans the existing (virtual) MAC queues, re-prioritize packets (by adjusting scheduling weights) and discard any stale data, and estimate the (maximal) available bandwidth on each radio for the current Hyper Scheduler scheduling period. Hyper Scheduler then polls the (tentative) list of user data radio assignment queues, and for each ("non-stale") data item, Hyper Scheduler assigns it to its most preferred radio if there is sufficient bandwidth, otherwise, the next preferred radio, and so forth. If no bandwidth is available on any radio, it is postponed for the next Hyper Scheduler scheduling period. The assigned data item is tagged with a scheduling weight and inserted into the corresponding low-level (virtual) MAC queue. The scheduling weights will be used by the low-level MAC scheduler for radio resource assignment.

Algorithm 1 Hyper_Scheduler_Stage_1_Procedure(mac_id)

```
packet_list ← RLC_list[mac_id].pdu_list
for packet ∈ packet_list do
    //loop through packets in RLC queue and in-place assign calculated tags
    //using HS table policy (destined radio, discard tag, ...)
    packet_qfi ← packet.tags.qfi
    packet_uid ← packet.tags.uid
    packet_UE_state ← packet.tags.UE_state
    assigned_policy ← policy_schedule_table_lookup(packet_qfi, packet_uid, packet_UE_state, radios_cqi)
    //Select the radio id based on the PD code in assigned_policy
    packet.tags.list_assigned_radio_channels ← select_radio_channels(assigned_policy)
    //Check for policy code and determine if packet should be discarded
    packet.tags.discard ← check_packet_discard(assigned_policy)
    packet.tags.bandwidth_requirement ← assigned_policy.bandwidth_requirement
end for
Hyper_Scheduler_Stage_2_Procedure()
```

Algorithm 2 Hyper_Scheduler_Stage_2_Procedure()

```
for queue ∈ mac_queue do
    //loop through in packet inside MAC virtual list, discard, delay, or process packet
    //Re-prioritize queue based on QFI and wait time
    queue ← HS_MAC_Prioritize(queue)
    packet_list ← queue.pdu_list
    for packet ∈ packet_list do
        list_assigned_radio_channels ← packet.tags.list_assigned_radio_channels
        discard ← packet.tags.discard
        bandwidth_requirement ← packet.tags.bandwidth_requirement
        if discard then
            continue
        else
            deliver ← False
            for radio_channel ∈ list_assigned_radio_channels do
                if list_radio_current_bandwidth[radio_channel] ≥ bandwidth_requirement then
                    MAC_send_packet(radio_channel, packet)
                    deliver ← True
                    break
                end if
            end for
            if not deliver then
                //Bandwidth of current radio does not suffice, delay until next sending round
                queue.append(packet)
            end if
        end if
    end for
end for
```

```yaml
UE:
  stationary:
    - conditions:
        - {greaterThan: {value1: "CQI_1", value2: 10}}
        - {greaterThan: {value1: "CQI_2", value2: 10}}
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
      action:
        - assignRadio: radioWWithMaxAvailableBandwidth
    - conditions:
        - {greaterThan: {value1: "CQI_1", value2: 10}}
        - {greaterThan: {value1: "CQI_2", value2: 10}}
        # for enhancement layer
        - {equalTo: {value1: "QFI", value2: 30}
          or {value1: "QFI", value2: 40}}
      action:
        - assignRadio: radioWithMinAvailableBandwidth
    - conditions:
        - {lessThanOrEqual: {value1: "CQI_1", value2: 10}}
        - {lessThanOrEqual: {value1: "CQI_2", value2: 10}}
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
      action:
        - assignRadio: radiowithMaxCQI
    - conditions:
        # for enhancement layer
        - {equalTo: {value1: "QFI", value2: 30}
          or {value1: "QFI", value2: 40}}
      action:
        - assignRadio: radioWithMinCQI
  walking:
    - conditions:
        - {greaterThanOrEqual: {value1: "CQI_1", value2: 5}}
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
      action:
        - assignRadio: radioWithMaxCQI
    - conditions:
        - {lessThan: {value1: "CQI_1", value2: 5}}
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
      action:
        - assignRadio: radiowithMaxCQI
    - conditions:
        # for enhancement layer
        - {equalTo: {value1: "QFI", value2: 30}
          or {value1: "QFI", value2: 40}}
      action:
        - assignRadio: radiowithMinCQI
  delivery_deadline:
    - conditions:
        - {greaterThan: {value1: "deadline", value2: 20}}
      action:
        - dropEnhancementLayer
```

Figure 5.3: Specification of Example Policy 1.

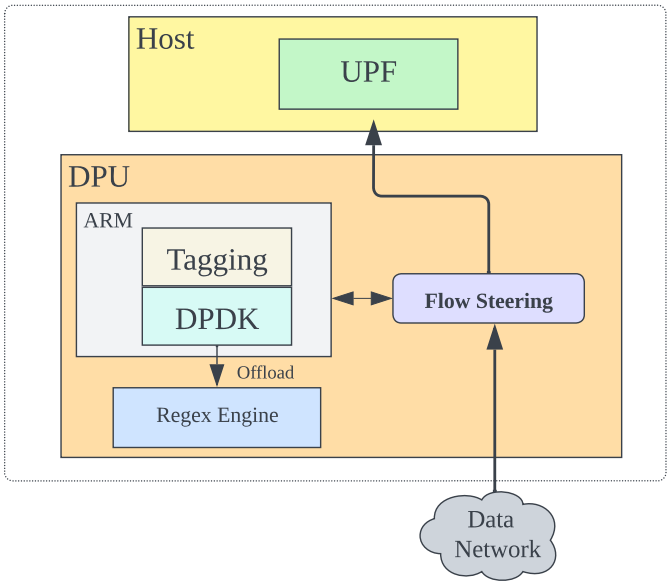
```
UE:
    stationary:
    - conditions:
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
      action:
        - assignRadio: radioWithMaxCQI
    - conditions:
        # for enhancement layer
        - {equalTo: {value1: "QFI", value2: 30}
          or {value1: "QFI", value2: 40}}
      action:
        - assignRadio: radioWithMinCQI
    - conditions:
        # for base layer
        - {equalTo: {value1: "QFI", value2: 10}
          or {value1: "QFI", value2: 20}}
        - notSufficientResource
      action:
        - assignRadio: BothRadios
    - conditions:
        # for enhancement layer
        - {equalTo: {value1: "QFI", value2: 30}
          or {value1: "QFI", value2: 40}}
        - notSufficientResource
      action:
        - assignRadio: drop
```

Figure 5.4: Specification of Example Policy 2.

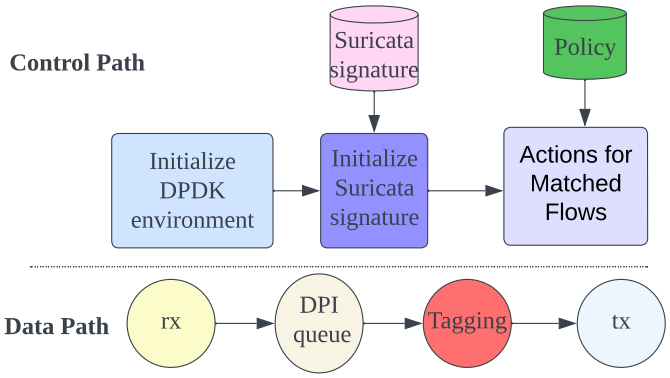
Packet Prioritization, Re-transmissions & Discarding. When a packet arrives from the UPF to O-CU, it may be marked with a timestamp as part of the new metadata (Fig. 5.8); this metadata may be passed down to O-DU with the packet. Depending on the scheduling policy/logic programmed for each user/service/data stream, Hyper Scheduler can use the timestamp to dynamically prioritize and schedule packets for transmissions/re-transmissions via the lower-level MAC radio schedulers. When data becomes stale (i.e., past its deadline), the Hyper Scheduler may discard it from the buffer. This avoids wasting scarce radio resources for unnecessary data transmissions/re-transmissions and ensures prioritized delivery of time-sensitive data.

5.4.3 HyperRAN Core Design

Fig. 5.6 illustrates the workflow on the core side of HyperRAN. In the first step, the application providers or operators define rules for the application traffic in terms of headers and traffic behaviors.

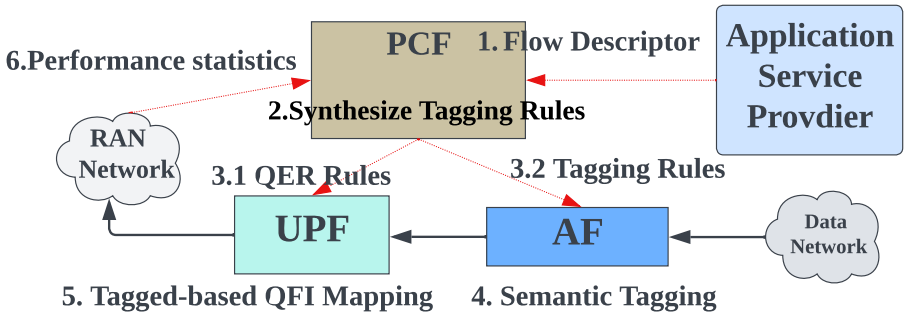


(a) Offloaded Operations

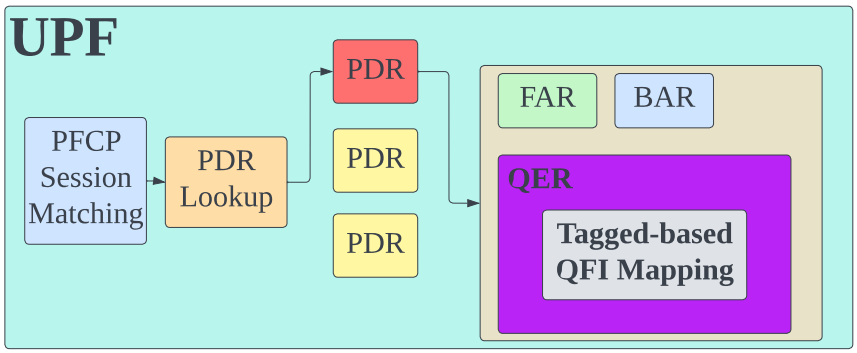


(b) DPU Initialization and Data-plane Operations

Figure 5.5: Offloading Tagging Operations to DPU.



(a) Semantic Tagging



(b) UPF

Figure 5.6: 5G Core Side Design.

In the second step, the PCF synthesizes rules based on all existing ones to create a more efficient representation to be deployed on the AF. The rules consist of two parts: the rules for tag-based QoS flow mapping (in the form of QER) for UPF and the ones for tagging the packets in the AF, which are installed in step 3. In step 4, the AF tags the packets based on headers and traffic characteristics, providing application-level insights to the packets. When the UPF receives the packets in step 5, it chooses the appropriate QFI value based on the semantic tags. If the O-RAN is unable to accommodate the application requirements, it reports the statistics to our controller for rule adjustment.

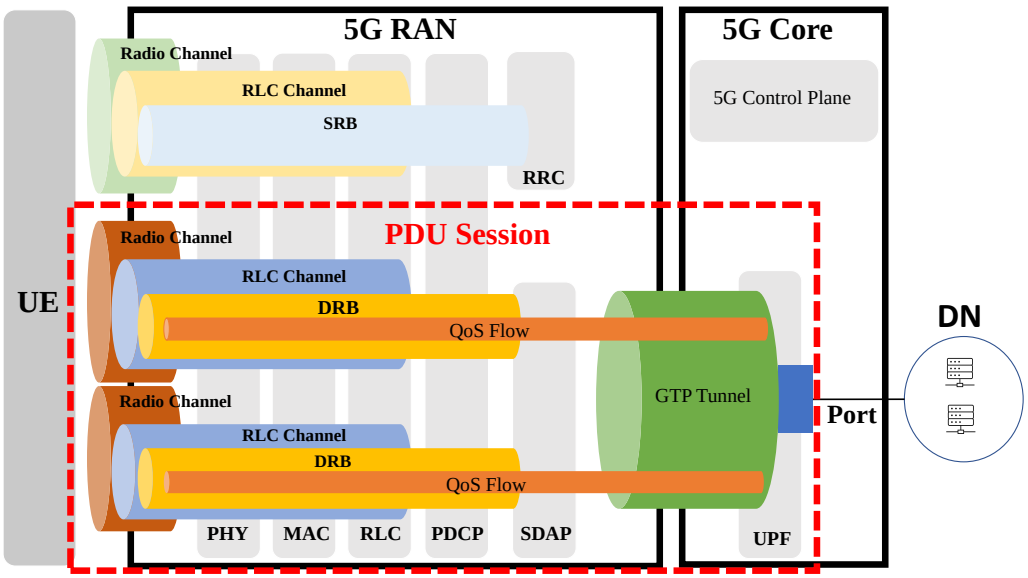


Figure 5.7: 5G RAN End-to-end Protocol Stack.

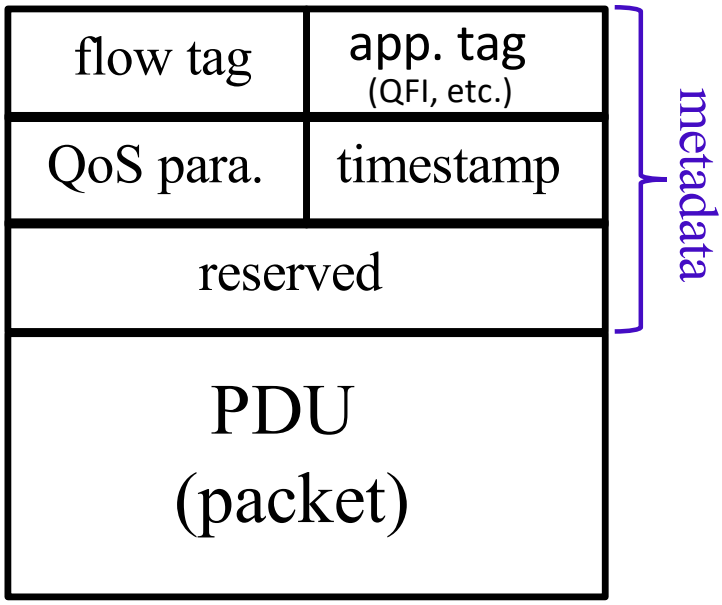


Figure 5.8: Packet Metadata.

5.5 Implementation and Experimental Setup

HyperRAN Implementation. We have implemented a prototype of HyperRAN on commodity Intel multi-core servers with 100 Gbps Ethernet NICs with the DPDK support. Our implementation relies on srsRAN [71] as the code-base, which incorporates CA, although we also frequently consult the OAI RAN implementation [55] for reference but neither srsRAN nor OAI RAN support multiple radios. We reuse the srsRAN's PHY/MAC layer implementation and set up multiple (slightly modified) PHY/MAC layers to emulate a multi-band, multi-RAT RAN. We completely re-architect the RLC/PDCP/SDAP layers, and modularize their implementations, reusing as much of the code as possible while re-implementing them into modular protocol functions (see §5.4). We implemented Hyper Scheduler in such a manner that it can be turned on or off during start-up or dynamically during runtime. We plan to make our implementation publicly available in the future.

5G Core, O-RAN & UE Implementations and Experimental Setup. To facilitate *end-to-end* evaluation, we also modify the implementation of Open5GS [85] which is used as the 5G core in our evaluation. Open5GS already supports QFI, we add a novel application semantics based QFI to the 5G core. We include the details of 5G core implementation in the §5.5.1. We leverage OAI's FlexRIC implementation [86] as the O-RAN RICs. We send the Hyper Scheduler's policies via the existing FlexRIC's E2 interface. We also extend the E2 interface in FlexRIC to update the Hyper Scheduler policies at the run-time. The policy operations include adding a new policy, modifying an existing policy, and deleting an existing policy. The Hyper Scheduler periodically sends statistics and other information to the FlexRIC. We further extend the implementation of srsRAN's UE (srsUE) to add multi-radio/multi-band support. We implemented multiple UEs support in HyperRAN to perform multiple clients and hybrid experiments. As discussed in §5.4.2, we implemented a WPS algorithm (more details in §5.5.2) in HyperRAN to support user data QoS requirements. We evaluate HyperRAN, and the entire 5G ecosystem (5G Core, RAN, O-RAN, Non-RT RIC, O-RAN Near-RT (nrt) RIC, etc.) as well as application use cases (see §5.6) on a testbed comprised of two Intel multi-core servers with 64 GB RAM and Core i5 processors running Ubuntu 20.04.

5.5.1 HyperRAN Core Implementation

As illustrated in Fig. 5.5, the tagging module is offloaded to the Nvidia Bluefield 2 [87] to potentially save CPU cycles on the host. By leveraging hardware-accelerated OVS, the packets can be directed to the ARM cores for tagging and statistics collection. The hardware-accelerated regular expression matching engine provided by the DPU can handle offloading pattern matching operations. To implement this functionality, we utilize the default ECPF (Embedded CPU Function Ownership Mode) [88] in the Bluefield.

The tagging module is implemented as a DPDK application [89]. In the control path, within the DPDK environment, fine-grained flow patterns are compiled into Suricata signatures, and tagging policies are initialized accordingly. In the data path, using Receive Side Scaling (RSS), the packets are distributed across multiple cores, each of which runs a polling thread. Upon receiving packets, the thread first checks if a flow ID is detected by the connection tracking hardware offloads. If the packets are not marked with a flow ID, new flows are created using the *doca_dpi_flow_create* [90]. Subsequently, the packets are inserted into a DPI queue for hardware-accelerated processing. By calling *doca_dpi_dequeue*, the application obtains the signature ID and utilizes it to perform the tagging operations defined by the tagging policy. Finally, the packets are forwarded to the host.

5.5.2 Weighted Proportional Sharing Algorithm

We have extended the MAC scheduler algorithm in srsRAN [71] to incorporate a WPS algorithm, enabling more precise and equitable resource allocation across UEs. This enhancement involves calculating DL and UL priorities, which are pivotal in the dynamic allocation of transmission time intervals (TTIs) to UEs. In the implementation, the DL and UL priorities for each UE are computed based on their expected bitrates r, the average network bitrate R, and a fairness coefficient f, all weighted by a specific value w assigned to each UE by the Hyper Scheduler, as described in §5.4.2. We illustrated the priority P as follows:

We evaluate the impact of weights on performance using two UEs, setting the f to the default value 2 and fix the value of w_2 to 1, illustrated in Table 5.2. According to

$$P = \begin{cases} \frac{r}{R^f \times w}, & \text{if } R \neq 0 \\ 0, & \text{if } r = 0 \\ \infty, & \text{otherwise} \end{cases}$$

Table 5.2: Impacts of Weights Configuration in WPS Algorithm

w_1/w_2	Throughput (UE1) [Mbps]	Throughput (UE2) [Mbps]
1	10.4	10.4
5	14.3	6.12
10	15.4	5.08
25	18.1	2.75
50	19.1	2.59

the result, the allocation of throughput is increased when we assign more weights to the UE.

5.6 Evaluation

We evaluate HyperRAN using three illustrative and representative application use cases that require high bandwidth and are time-sensitive: 1) volumetric video streaming, 2) LiDAR streaming (for *cooperative* autonomous driving) and 3) hybrid experiments using both video streaming, and LiDAR streaming with multiple UEs. We use an experimental set-up presented in §5.5 where HyperRAN is equipped with two radios, each configured with two channels. For baseline experiments, we employ CA which uses single radio with four channels. For end-to-end evaluation, we implement a video streaming simulator and LiDAR streaming pipeline. Additionally, we added End-point Adaptation support to video streaming simulator using ABR algorithms for additional comparisons. This algorithm aims to occupy maximum bandwidth by adjusting bitrate according to channel conditions. We evaluate the performance of HyperRAN using different Hyper Scheduler policies and compare them with baselines. For "realistic" evaluation, we collect real-world multi-radio/channel bandwidth and CQI traces from

Figure 5.9: Data Collection.

commercial 5G networks to drive the emulations. All evaluations are conducted under the same 5G network trace files emulating a consistent environment ensuring fairness of comparisons. We expect our results to translate well to real 5G speeds and latency.

Commercial 5G Multi-Radio Datasets. Fig. 5.9 illustrates our 5G data collection platform, where smartphones are placed side-by-side and tethered to the professional 5G diagnostic tool XCAL [91]. We performed two experiments, with each phone locked to a different radio channel serving as the baseline and employing the same default channel for the multiple UEs competition. We collected PHY throughput and CQI data of four 5G mid-band channels from two US 5G operators under various environments and mobility settings (stationary, walking, and driving), see Table 5.3 for summary trace statistics. Fig. 5.10 show sample CQI traces of 4 channels collected at the same time. We see that the channel conditions can dramatically change, and no channel is consistently better than the others.

Table 5.3: Data Trace Statistics.

Commercial 5G Traces

Mobility	Stationary	Driving
Scenario	Downtown	Highway Downtown
Duration	∼60 min	∼430 min
Coverage	12 hot spots	∼650 km
# of Operators	2	2
# of Channels	4	4

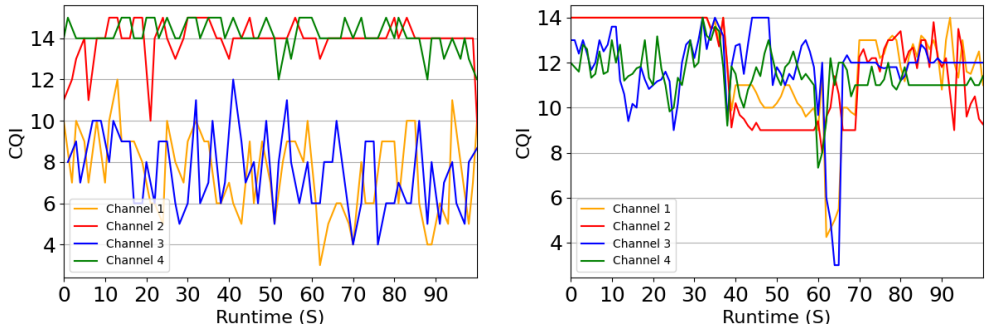


Figure 5.10: Sample Time-series Plots of CQIs under Stationary (left) and Driving (right) Settings.

5.6.1 Prioritizing Volumetric Video Layers to Reduce User Perceived Stall Time

Application Semantics. In volumetric video streaming, each frame is segmented into multiple layers. The number of layers successfully delivered to the client side affects the quality of the frame in the video streaming client. The first two layers, or base layers, are required to play a frame at a minimal quality, and thus their delivery is most important. Each subsequent layer above the base layers referred to as enhancement layers, rely on strict layer order being delivered or else be discarded by the client and thus render a lower-quality frame and/or cause stalling. Thus, the Hyper Scheduler will prioritize the lower layers based on the data's QFI value, each channel's bandwidth, and CQI. With this approach, the base layers are prioritized. In the context of our experiments, Hyper Scheduler may decide against sending enhancement frames to preserve the QoE of the video playback if the network is particularly constrained.

Table 5.4: Video Stall Time (Stationary).

Points/Frame	4K	5K	6K
Required Tput (Mbps)	86.4	108	129.6
CA	37s	59s	71s
End-point Adaptation	25s	28s	34s
Static Channel Mapping	20s	25s	30s
HyperRAN EP1	12s	13s	17s
HyperRAN EP2	10s	13s	16s

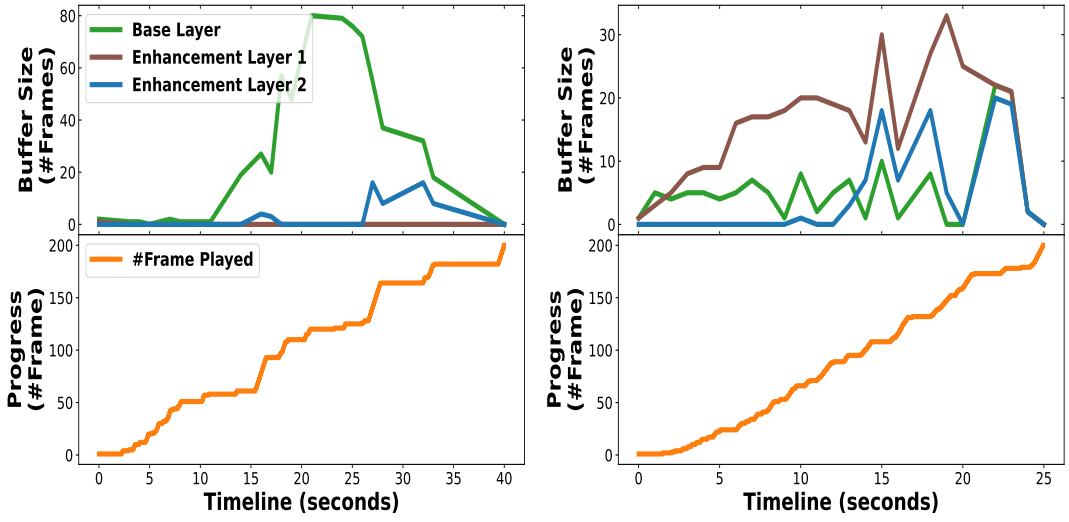
Table 5.5: Video Stall Time (Walking).

Points/Frame	4K	5K	6K
Required Tput (Mbps)	86.4	108	129.6
CA	124s	128s	133s
End-point Adaptation	72s	76s	98s
Static Channel Mapping	50s	60s	69s
HyperRAN EP1	41s	48s	49s
HyperRAN EP2	40s	48s	50s

Volumetric Video Baseline Scenarios and Metrics Collected. We evaluate two Hyper Scheduler policies EP1 and EP2 in the Volumetric Video streaming scenario against a case where video is delivered with CA, where Static Channel Mapping assigns specific QFI marked packets to a specific radio channel, and where an End-point Adaptation algorithm adjusts streaming bitrates in face of network constraints. We also evaluate a multiple UEs case with two simultaneous devices running video streaming. We collect two metrics: (1) Stall time, the amount of time a stream spends waiting for data to come in and (2) Buffer size, the amount of frames in the client buffer awaiting playback.

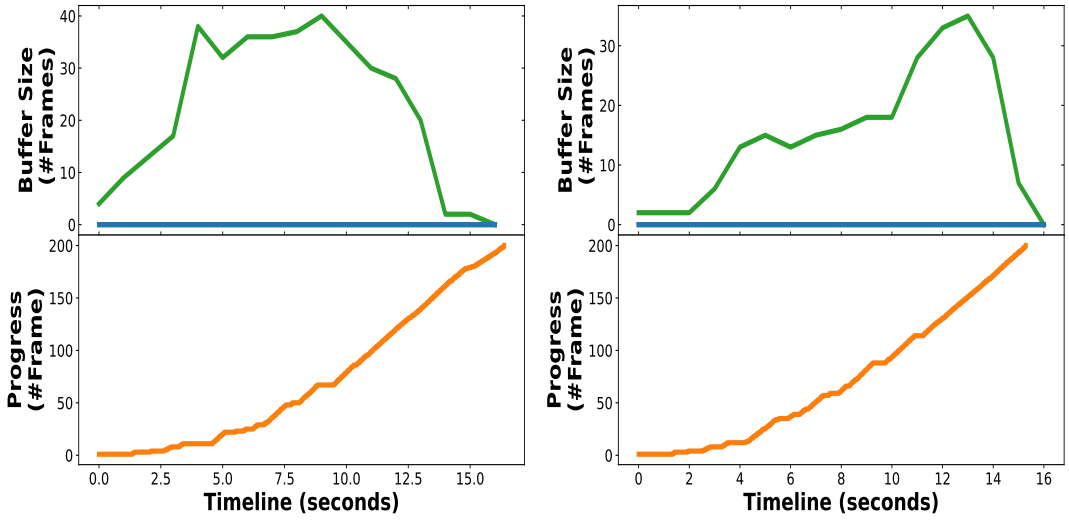
Table 5.6: Video Stall Time (Driving) - Multiple UEs- Hybrid Experiments.

Points/Frame	4K	5K	6K
Required Tput	86.4	108	129.6
Static Channel Mapping	179s	213s	280s
HyperRAN EP2	65s	82s	95s



(a) CA

(b) Static Channel Mapping



(c) HyperRAN EP1

(d) HyperRAN EP2

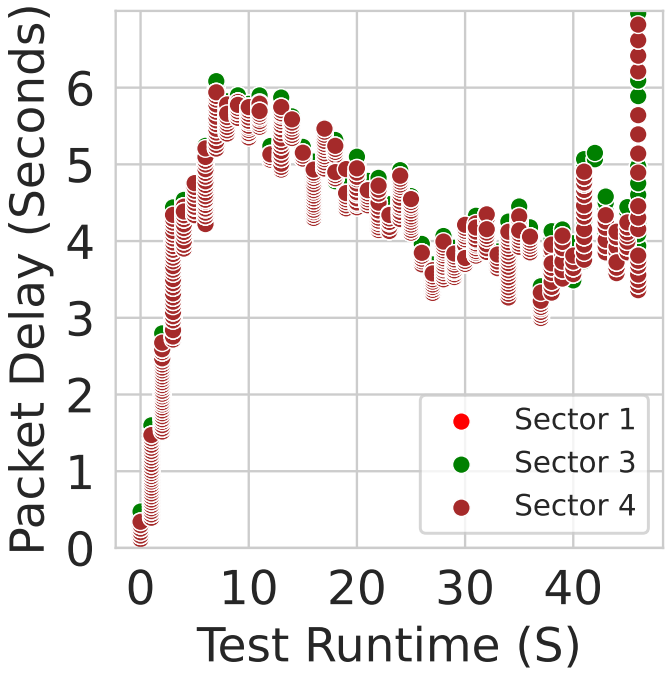
Figure 5.11: QoE Performance of 4K Points/Frame with Video Player Progress, Video Frame Quality Metrics (Stationary). Upper subfigures report buffer size for each layer during transmission, and lower subfigures report the number of frames sent as the progress. The lower buffer size and shorter total transmission time indicate better performance.

Results. Our evaluations are outlined in Tables 5.4, 5.5 and Fig.5.11. From the experimental results, we see that the CA case has the highest buffer size, indicating failures to deliver the base layers (refer Fig. 5.11a), resulting in frequent stalls averaging a total of 55 seconds. Fig. 5.11b shows that the Static Channel Mapping case fairs a little better utilizing both radios to effectively deliver more layers, but due to enhancement layers competing for radio resources with the base layers, the overall stall time is still quite high at an average of 25 seconds. As shown in Table 5.4, EP1 and EP2 provide a dramatically lower stall rate averaging 13 seconds, yielding a 46-75% reduction in stall time compared to the baselines. Notably, despite the application-level adaptation occurring in the End-point Adaptation approach, HyperRAN's RAN level optimizations provide better performance.

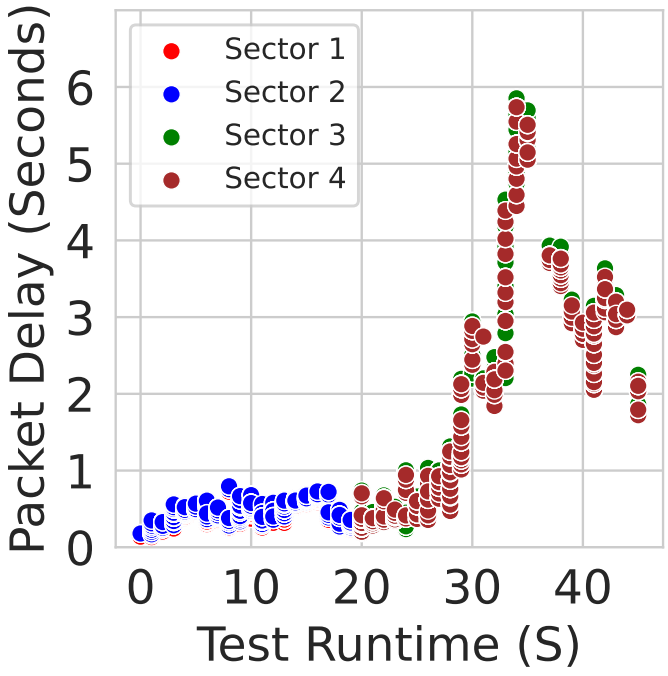
Considering the mobility cases, despite the more challenging mobility of driving, we see in Table 5.5, HyperRAN provides a 22-64% improvement to stall time over the baselines. This is even more noticeable under the multiple UEs assumption where HyperRAN EP2 results in a 63% reduction in stall time. Another contrast, the CA case is unable to deliver enhancement layers or base layers effectively due to poor utilization of radio resources. As shown in Figs. 5.11c & 5.11d, HyperRAN achieves the best overall performance by dropping enhancement layers under bottleneck and prioritizing base layers delivery. This is visualized in the tighter spread of stall times observed in Fig. 5.14, where the CA case sees a wide range and overall worse performance for base layers and enhancement layers contrasted with the enhancement-less EP1 and EP2 cases.

5.6.2 Prioritizing Context Important LiDAR Data through Smart Partitioning

Application Semantics. To leverage HyperRAN for the LiDAR use-case we identify application semantics needed for streaming. LiDAR is composed of many point samples generated by a full 360-degree sweep around a device. This constitutes a large amount of data to transmit, and not all of this is equally important at all times. From a Connected Autonomous Vehicle (CAV) example with the LiDAR scan broken up in Fig. 5.15, a vehicle may be spotted only in sector Q1 & Q2, and thus be critical data to send leaving the other zones at a lower priority. We annotate the LiDAR data with priority

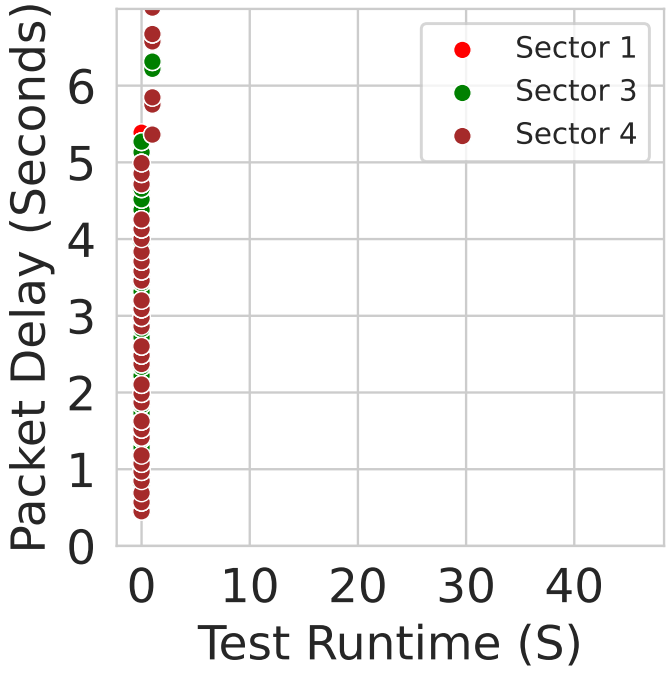


(a) Static Channel Mapping.

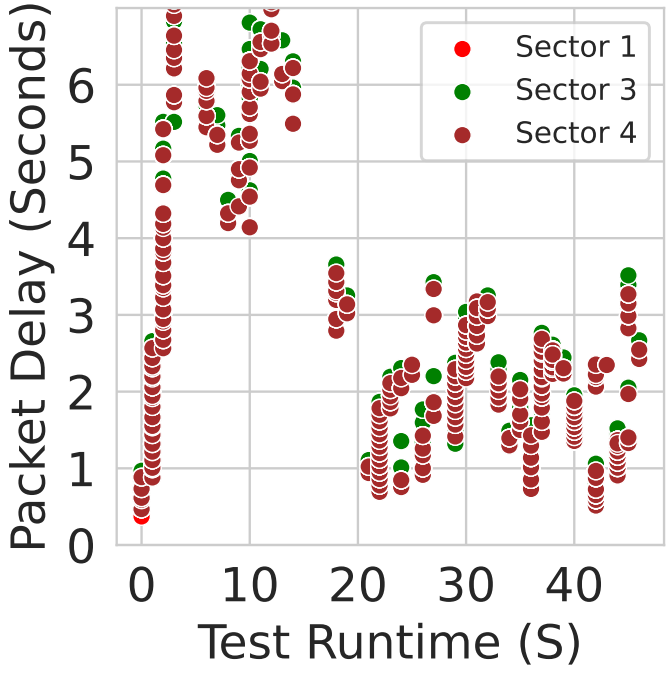


(b) HyperRAN EP2 QFI-based Dynamic Mapping.

Figure 5.12: Baseline vs HyperRAN for QFI Timing (Stationary).



(a) Static Channel Mapping



(b) HyperRAN EP2

Figure 5.13: Multiple UEs, LiDAR and 6K Video (Driving).

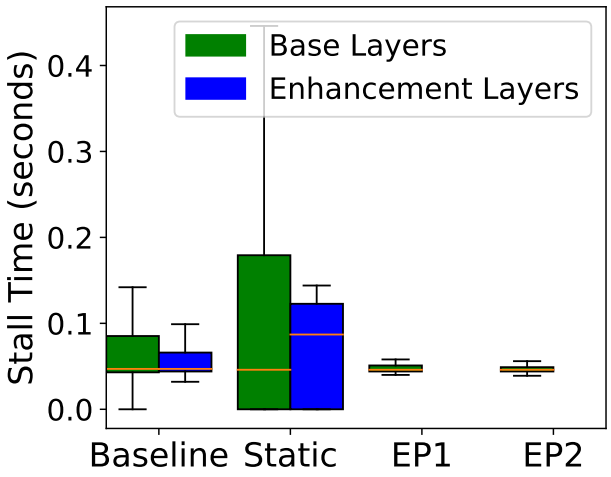


Figure 5.14: Stall Time of Base & Enhancement Layers of 4K Points/Frame (Stationary).

semantics and HyperRAN will deliver each partition's LiDAR data with Q1 & Q2 at highest priority. The priority attached to Q1 & Q2 could migrate to Q3 and Q4 as the sighted vehicle is detected in those subsequent sectors, resulting in lower priority for the now empty Q1 & Q2. HyperRAN may also discard data in sectors that are delivered too late, as in the real use-case samples where a vehicle was is not as important as where it currently is. Additionally, in the creation of these semantics we leverage HyperRAN's multi-radio support to send the top priority sectors on two different radios along with the lower priority sectors reducing competition for radio resources.

LiDAR Baseline Scenario and Metrics Collected. We evaluate against a simple Static Channel Mapping (SCM) case, providing a baseline and contrast for HyperRAN's ability to deliver semantically marked traffic over a diverse radio and condition environment. We utilize two test cases: (1) Like the example described above, two vehicles enter the radius of the detecting vehicle, resulting in the need to prioritize the sectors where the vehicles are and change priority as the vehicle moves around the LiDAR. The priority order of vehicle 1 is 8 & 7 then 6 & 5. Vehicle 2's order is 1 & 2 then 2 & 3. (2) We evaluate a multiple UEs case wherein volumetric video streaming at differing quality rates is occurring alongside LiDAR streaming on two separate devices going through the same RAN. This illustrates how HyperRAN leverages WPS to improve delivery of

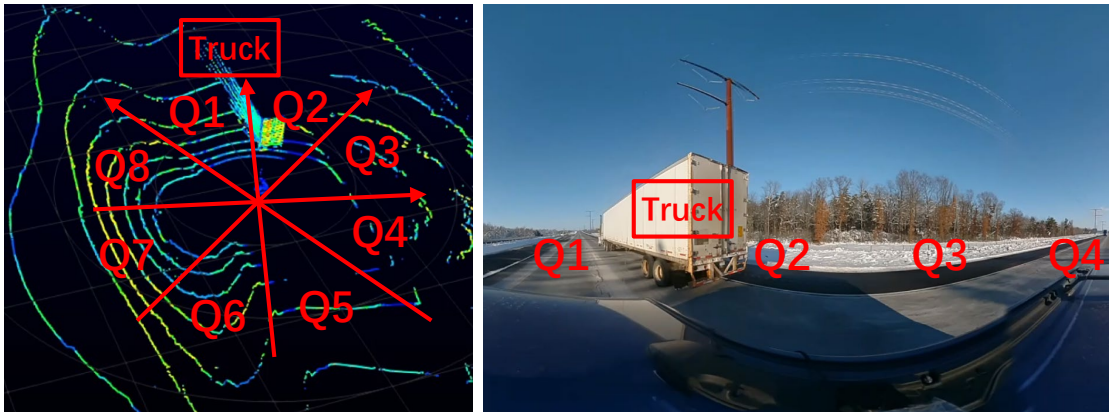


Figure 5.15: LiDAR and Sectors Reference.

Table 5.7: LiDAR Metrics. Priority Swapping Tests. (Stationary)

Approach	Sector	Avg Sample Latency (S)	Avg Sample Loss %	Sector	Avg Sample Latency (S)	Avg Sample Loss %
Static Channel Mapping	**Sector 1**	0.85 ± 0.07	99.90%	**Sector 5**	6.19 ± 2.17	64.66%
	Sector 2	N/A ± N/A	100.00%	**Sector 6**	5.53 ± 2.25	57.93%
	Sector 3	4.22 ± 1.20	56.50%	**Sector 7**	3.53 ± 1.30	90.52%
	Sector 4	4.13 ± 1.20	55.24%	**Sector 8**	4.22 ± 0.76	98.02%
HyperRAN EP2	**Sector 1**	0.46 ± 0.13	63.18%	**Sector 5**	1.55 ± 0.75	74.45%
	Sector 2	0.42 ± 0.13	62.28%	**Sector 6**	1.49 ± 0.75	74.89%
	Sector 3	2.01 ± 1.54	76.33%	**Sector 7**	0.66 ± 0.48	66.84%
	Sector 4	1.93 ± 1.48	76.24%	**Sector 8**	0.70 ± 0.48	66.38%

LiDAR data under multiple client's pressure. There are no changing priorities for this use-case and Sectors 3, 4, 5, and 6 are set as high priority. From our experiments we collect (1) Packets Discarded by Policy, which reflects the number of packets the HyperRAN policy drops during operation, (2) Time Packet in Queue, which reflects how much time a packet held by the RAN is queued before being delivered to the destination, (3) Sample Latency, which is the application-level time it takes for a single LiDAR sample (made up of several packets) to arrive on the destination side, and (4) the Sample Loss, which is how many LiDAR samples were lost vs those sent by the client application.

Results. In Table. 5.7, we see the baseline SCM struggle to consistently and quickly deliver data packets resulting in latency roughly 2x+ slower than the HyperRAN results. We also see HyperRAN's discard policy helping to lower the overall packet loss by around 7.7% across all sectors. Similarly, this discard and priority approach overcomes the significant sector failure the baseline suffered with Sectors 1 and 2. With further

Table 5.8: RAN Metrics, Multiple UEs 6K Video Streaming (Driving).

Approach	QFI	Time Packet in Queue (S)	Packets Discarded By Policy
HyperRAN EP2	**11**	1.18± 0.80	0.00%
	12	2.71± 2.27	0.00%
	13	3.82± 2.38	99.89%
	14	7.77± 6.77	99.69%
Static Channel Mapping	**11**	6.35± 2.36	0.00%
	12	11.82± 4.31	0.00%
	13	5.84± 2.18	0.00%
	14	12.58± 4.48	0.00%

examination of the SCM case in Fig. 5.12a, we see failure to deliver virtually all of Sector 2 & 1's packets. In contrast, in HyperRAN's Fig. 5.12b, we see prioritization delivers Sector 1 & 2 with lower loss rates and faster speeds, and then swaps priority to Sector 3 & 4 for the second half. **Note:** Sectors 5-8 omitted from figures for clarity. From the second test case involving multiple UEs and parallel video streaming, we see that HyperRAN provides a beneficial triage-like effect under network competition. In Figs. 5.13a-5.13b HyperRAN provides a 2-4x reduction in latency for the higher priority sectors (more in Table 5.9). This priority is reflected in the 100% (or near) loss rates in the lower priority sectors as HyperRAN is discarding lower priority data to make room for the LiDAR data amongst the video data pressure. This impact is reflected for RAN metrics in Table. 5.8 where we see overall loss is shifted onto the lower priority QFI values providing lower 15% lower loss rate and faster speeds. **Note:** The video and LiDAR share the same prioritization, and thus when grouped by QFI will contain values differing somewhat from the LiDAR specific results. **Note:** LiDAR sends lots of UDP data and the constrained nature of our testing system results in high loss such as the 90% observed in baseline.

5.7 Related Work

Since the late 90's [92, 93], scheduling algorithms in wireless data networks that take into account channel conditions, diversity and fairness while providing QoS to users have been studied extensively; voluminous theoretical studies have been published, see, e.g., [94, 95, 96, 97, 98, 83, 99, 100] and survey paper [101]. Measurement-based and theoretical analysis of its effectiveness in these networks have been carried out, see,

Table 5.9: LiDAR QoS and QoE Metrics, Multiple UEs Tests alongside Volumetric Video Streaming.

Approach	Sector	Avg Sample Latency (S)	Avg Sample Loss %	Sector	Avg Sample Latency (S)	Avg Sample Loss %
HyperRAN EP2 - 4k	**1**	nan ± nan	100.00%	**5**	1.77 ± 0.71	65.33%
	2	nan ± nan	100.00%	**6**	1.65 ± 0.68	66.30%
	3	3.10 ± 1.58	68.26%	**7**	nan ± nan	100.00%
	4	2.94 ± 1.57	67.02%	**8**	1.36 ± 0.20	99.95%
Static Channel Mapping - 4k	**1**	14.61 ± 3.06	94.78%	**5**	7.54 ± 1.52	84.80%
	2	14.54 ± 1.17	98.93%	**6**	7.01 ± 1.94	82.56%
	3	12.38 ± 4.75	83.16%	**7**	5.94 ± 3.18	91.78%
	4	12.80 ± 4.61	79.91%	**8**	6.07 ± 3.40	94.27%
HyperRAN EP2 - 6k	**1**	0.57 ± 0.56	99.80%	**5**	2.00 ± 0.81	64.80%
	2	nan ± nan	100.00%	**6**	1.93 ± 0.81	65.94%
	3	2.95 ± 1.81	70.27%	**7**	nan ± nan	100.00%
	4	2.86 ± 1.82	69.59%	**8**	nan ± nan	100.00%
Static Channel Mapping - 6k	**1**	9.13 ± 5.18	99.83%	**5**	3.89 ± 1.96	99.58%
	2	nan ± nan	100.00%	**6**	5.08 ± 2.72	93.75%
	3	12.94 ± 4.46	79.59%	**7**	6.83 ± 2.34	78.12%
	4	12.71 ± 4.35	78.33%	**8**	7.52 ± 2.01	82.96%

e.g., [102, 103, 82, 104]. None of these studies consider incorporating application semantics in scheduling decisions. Cross-layer design has been a major theme in wireless networks, but most studies largely rely on passing relevant information up or down the protocol stack to address specific problems, e.g., congestion control and related issues [105, 106, 107, 108, 109, 110, 111, 112]. Network slicing is another topic that has been widely studied, see [113] for examining the use of network slicing to support QoS in 5G networks. Machine learning has been widely applied to tackle various problems in wireless networks, e.g., for spectrum monitoring, channel modeling and estimation, massive MIMO and beam-forming, power control, user detection and mobility tracking, and so forth (see, e.g., [114, 115, 116, 117, 118]; and [119, 120, 121, 122, 123] for surveys). None of these studies tackle the fundamental limitations of the current 5G network architecture. The two available open source RAN projects OAI [55] and srsRAN [71] doesn't support multiple-bands simultaneously. We first highlight the limitations of 5G flow-based QoS architecture in an earlier *position* paper [124] and *prototype* paper [125].

5.8 Summary

In this chapter, we designed, implemented, and evaluated HyperRAN, a fine-grained, cross-edged, QoS framework that provides an enhanced QoE for emerging applications.

HyperRAN is the first framework that supports emulation of multiple channels simultaneously and excels in performance when compared to other available open-source RAN systems. The Hyper Scheduler is integrated with the O-RAN RIC and send metrics. Our system provides significant stability and improved QoS and QoE for relevant use-cases in volumetric video and LiDAR streaming. In the future, we plan to employ machine learning algorithms on the collected data to more efficiently use the radio resources. We would also like to exploit the LiDAR data and perform packet-based QoS approach to efficiently prioritize critical data and to drop/de-prioritize irrelevant/lower priority frames. Finally, we hope to examine real-time smart prioritization of multi-modal data streams.

Chapter 6

PRANAVAM: Scaling Private 5G RAN via eBPF+XDP

6.1 Introduction

With the needs for more flexibility, openness and programmability, not only cellular core networks but also radio access networks (RANs) are moving towards virtualization and cloudification. Both 3GPP and the Open-RAN (O-RAN) Alliance have introduced new *disaggregated* RAN architectures that divide 5G RANs into, for example, *Central Unit* (CU) and *Distributed Unit* (DU). CU is further split into CU-UP (CU user plane) and CU-CP (CU control plane), see Chapter 2 for more details. In particular, the O-RAN Alliance has introduced intelligent RAN controllers (RICs) and defined open interfaces for communications among the disaggregated units and RICs. Softwarization or "cloudification" of 5G and Next-Generation (NextG) RANs and core networks are especially appealing to many industrial use cases, as it makes it easier to support industrial verticals and *private* 5G [126] and NextG networks For example, RAN functionality can be tailored to the specific bandwidth, latency and reliability requirements of these use cases, and existing features may be upgraded or new features can be readily rolled out as the requirements or use cases change over time.

While software affords the benefits of programmability and scale-out, software implementation of RAN is in generally far slower than dedicated hardware appliances. This is further compounded by the needs for more complex, dynamic and intelligent features

in NextG RANs. This is particular the case for applications that that require a large number of simultaneous connections, high bandwidth, low latency and stringent reliability such as many industrial IoT (Internet of Things) and Industrial 4.0 Digital Twins use cases. Therefore, *scaling the NextG RAN software architecture while maintaining its programmability and openness is a key challenge in future RAN development.*

In this work we advocate an eBPF+XDP-based framework for scaling and accelerating software packet processing in NextG RANs. As a concrete example, we focus on 5G CU-UP as a key case study. On the one hand, CU-UP performs the upper layers of the 5G RAN protocol stack – Service Data Adaptation Protocol (SDAP) and Packet Data Convergence Protocol (PDCP) – and does not require specialized radio signal processing hardware (see Chapter 2). On the other hand, CU-UP often connects with several UPFs (User Plane Functions) in the 5G core as well as multiple DUs. As it lies on the critical path between the users and service endpoints, it must be capable of processing 10s or 100s millions of downlink packets from the core network to user equipment (UE) and uplink packets from UE to the core network per second in order to meet the bandwidth demands and minimize latency. Taking advantage that connections between CU-DU and CU-UPF are Ethernet-based, we exploit exploit eBPF and XDP for kernel extension, kernel bypassing and software packet processing optimization. We summarize the **key contributions** of our chapter below.

• We present an eBPF+XDP-based framework, dubbed PRANAVAM, for (O-RAN compliant) future RAN architecture development. Using 5G CU-UP as a key case study, we outline the initial design of our proposed PRANAVAM.

• Using eBPF+XDP for kernel extension/bypassing, our preliminary evaluation shows that PRANAVAM improves the throughput by 22-26% over existing 5G RAN implementations. We will make our code publicly available.

• We also discuss an additional design, PRAVEGA, in which the GTP-U packets are completely handled in the kernel space without being sent to the user space.

• We also discuss additional options to further accelerate software packet processing to scale 5G RAN implementation to meet bandwidth and latency demands.

While our initial design focuses on 5G RAN CU-UP, the ultimate goal is to apply PRANAVAM as a general framework for future RAN architecture development to meet the needs for openness, programmability, scalability and evolvability.

6.2 Design

In this section, we present PRANAVAM's architecture and design for fast processing of user plane packets in CU-UP of the RAN. The overall design of PRANAVAM is schematically sketched in Fig.6.1.

Features. The key feature of CU-UP is *routing* and *forwarding* packets to respective DUs in downlink or UPFs in uplink. The main features supported by PRANAVAM are listed below:

eBPF Maps. There are several types of eBPF Maps [127] available and each map is used for a particular purpose. In our design, we use two types of eBPF Maps. First, the *"eBPF XSKMAP Map"*, which is used to redirects raw XDP frames to AF_XDP sockets (XSKs) and it has two ring buffers, *"RX_Ring"* and *"TX_Ring"*. Second, *"eBPF PERCPU_ARRAY Map"*, which is used to store and retrieve traffic statistics between the kernel and user space. More details about how these eBPF Maps are used are discussed in detail in the below sections.

Design. As shown in Fig.6.1, PRANAVAM is divided into three layers: (i) Management Layer, (ii) Data Path Kernel Layer (DPKL), and (iii) Data Path User Layer (DPUL). During uplink or downlink, the user plane data passes through both DPKL and DPUL. The packets are processed differently during uplink and downlink. We will discuss the detailed design of each of these layers below.

6.2.1 Management Layer

This is one of the user space layer which manages the control plane part of PRANAVAM. The Management Layer consists of three main components as listed below:

(i) **E1 Session Manager.** It actively manages the E1 Session of CU-UP with CU-CP. During start-up, the CU-UP connects to the configured CU-CP. The CU-UP can connect to only one CU-CP. For each connected UE, the CU-CP instructs CU-UP with necessary information about the UE which includes QFI to DRB mapping and along with the ciphering and integrity protection for each DRBs. The E1 Session Manager instructs the eBPF Program Manager to dynamically load the eBPF program into the kernel space.

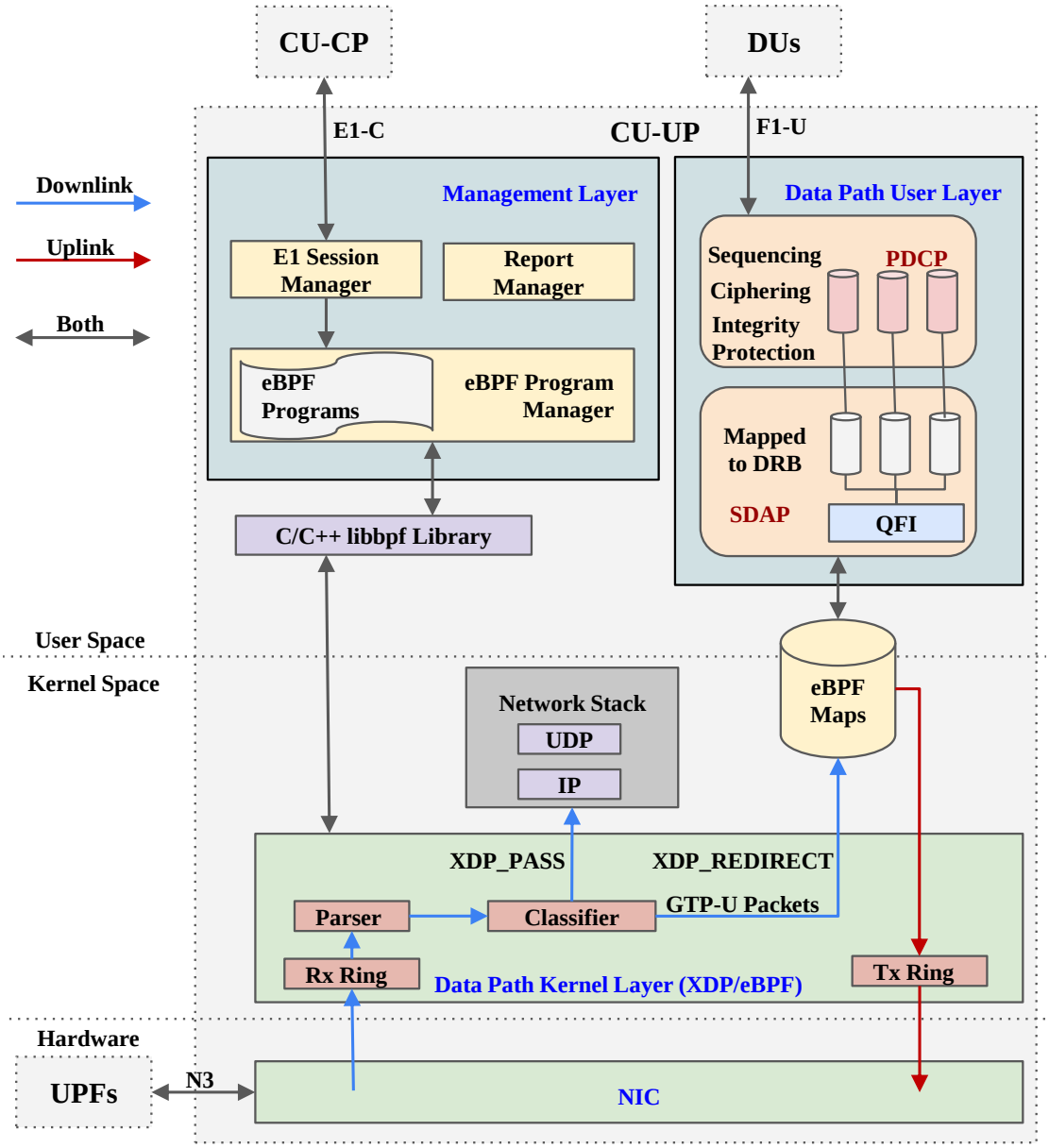


Figure 6.1: PRANAVAM- eBPF/XDP Socket Based CU-UP

(ii) **eBPF Program Manager.** It is responsible to manage the lifecycle of the eBPF program. Based on the instruction from the *E1 Session Manager*, it loads the eBPF bytecode in the kernel space. There will be only one eBPF program per NIC. If

the CU-UP machine has multiple NIC, then the *eBPF Program Manager* loads a eBPF program for each NIC.

(iii) **Report Manager.** It manages the status and statistical information of the components of PRANAVAM. It interacts with the *"eBPF PERCPU_ARRAY Map"* to get the sent and received packets information in the DPKL periodically. It also gets the sent and received packets information in the DPUL as well using an API.

6.2.2 Data Path Kernel Layer

This is the kernel space layer which process the user plane traffic inside the eBPF/XDP and is one of the Data Path layers. The DPKL uses *"eBPF XSKMAP Map"* to send (downlink) and receive (uplink) packets directly to and from the DPUL.

Downlink During downlink, the CU-UP receives the user plane data from the UPFs. Following are the key components of DPKL for downlink:

(i) **Parser.** This is the first component which gets called for each packet (from the UPF) received in the NIC. The main functionality of the *Parser* is to parse each packet and validate it for valid *"ethernet header"* structure. If the packet doesn't have valid structure, then the packets are dropped using *"XDP_DROP"*. The *Parser* component passes the valid packets to the *Classifier*.

(ii) **Classifier.** It classifies and send downlink GTP-U packets to DPUL using *"XDP_REDIRECT"* through *"eBPF XSKMAP Map"* and pass other packets to the network stack using *"XDP_PASS"*. The classifier uses two fields each GTP-U packet to classify downlink GTP-U packets. The first field is *"Message Type"* and the value should be "255" for GTP-U PDU Sessions. And the next field is *"PDU Type"* in the *"PDU Session Container"* extension header of the GTP-U packet which should be "0" for downlink. The value will be "1" for uplink GTP-U packets. When a GTP-U packet contains any other value in either of these fields, then the packet will be sent to the network stack rather than "eBPF XSKMAP Map".

Uplink There is no work in kernel space for uplink because the GTP-U packets are already parsed, classified and inserted directly into the *"eBPF XSKMAP Map"*s *"TX_Ring"* and assigned to the kernel space in the user space itself, which sends the packets to the UPF through the NIC.

6.2.3 Data Path User Layer

DPUL is the other user space layer and the other part of the Data Path layer. This layer consists of the higher layers of the RAN protocol stack such as PDCP and SDAP.
Downlink During downlink, the packets from the DPKL are available in *"eBPF XSKMAP Map"*s *"RX_Ring"*. DPUL polls the *"eBPF XSKMAP Map"*s *"RX_Ring"* for packets. First, each packet is processed for SDAP function in which the QFI value is retrieved and respective DRB is assigned in the PDCP function based on the QFI to DRB mapping. Next, the data undergoes the configured PDCP function processing such as integrity protection and ciphering. Finally, the data is routed to the respective DU based on the established PDU session.
Uplink In the current design, *eBPF* is not used for data data traffic between CU-UP and DUs. During uplink, DPUL receives the user plane data from the DUs using regular socket, then process the PDCP function and assign the data to the respective DRBs in which integrity protection and ciphering are done. Next, the SDAP function is processed and the respective DRB to QFI mapping is assigned. Finally, instead of routing the packet to UPF using regular socket, the data is inserted into the *"eBPF XSKMAP Map"*s *"TX_Ring"*. In order to send the data to the UPF, the kernel need to be explicitly notified using the *"sendto()"* call.

6.3 Implementation

In this section, we briefly discuss the implementation of PRANAVAM's design discussed in §6.2. For CU-UP, we considered to leverage either srsRAN [71] or Open Air Interface (OAI) RAN [55]. We decided to leverage OAI because OAI supports the O-RAN split using multiple network interfaces. Even though srsRAN claims that they support O-RAN support, their CU-CP, CU-UP and DU are integrated with tightly-coupled function calls and not network interfaces. The Management Layer and DPUL are developed in C and the DPKL is developed using restrict C. As discussed in the design section, we implemented two types of eBPF Maps and the *"eBPF XSKMAP Map"* has two ring buffers. Not all network interfaces support XDP (native mode), so we added support for both generic and native mode. In generic mode, PRANAVAM will work continue to work without any issues but there won't be any improved performance as seen in native

mode. In order to avoid packet loss during heavy traffic and for better performance since we run the application and the driver on the same core, PRANAVAM supports *poll* mode. Even if we run the CU-UP and the kernel driver in different cores, *poll* mode reduces the number of *syscalls* needed for TX path.

6.4 Preliminary Evaluation

6.4.1 Test Setup

The test setup for PRANAVAM's performance evaluation is shown in Fig.6.2. We conducted the experiments in a 6 CPU and 8GB RAM Ubuntu 20.04.6 OS virtual machine created on top of 11th Gen Intel(R) Core(TM) i3, 4 Cores, 8 logical processors machine. The end-to-end 5G system which consists of 5G core, RAN and UE was emulated in a single machine. We used OAIs 5G core, CU-CP, DU, UE along with our modified CU-UP for our experiments. Each component (AMF, SMF, NRF, UPF) of the OAIs 5G core were running in its own docker container inside the virtual machine. The CU-CP, CU-UP, DU and UE were ran as processes inside the virtual machine. We didn't isolate the system under test on purpose because we want to show the end-to-end performance results under close to realistic conditions.

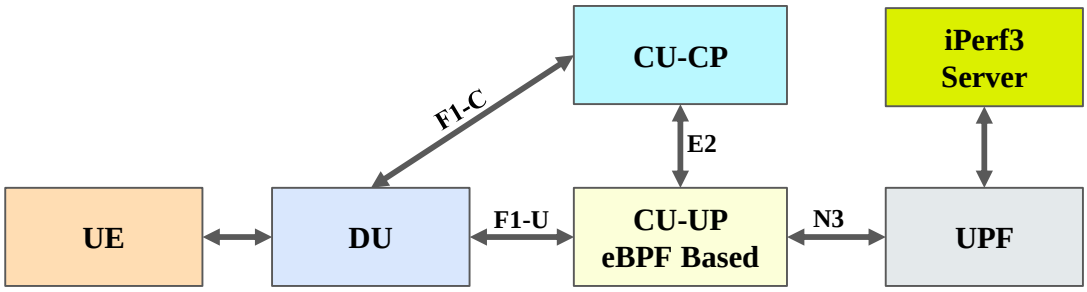


Figure 6.2: Test Setup

6.4.2 Data Traffic Generation

For evaluation, we used iperf3 [128] as a data generator. Our focus was on comparing the performance between regular socket and eBPF/XDP socket. We didn't use high-speed traffic generators because the other OAI components used in the end-to-end tests

don't achieve higher throughput. We ran the iperf3 server in a server machine behind UPF and the client in the OAI UE. In the client side, we used -P option to generate multiple parallel flows for generating diverse traffic, -u for UDP packets and -R option to conduct performance evaluation in the downlink direction.

6.4.3 Results

Fig.6.3 shows the throughput per number of parallel connections from a single UE. The results shows a performance improvement of around 22-26 percent in PRANAVAM in the downlink direction from the 5G core to UE. There is a linearity between the number of parallel connections and the throughput. We also noticed that there is a performance degrade in both regular socket and XDP as the number of parallel connection increases. We have shown the results only for the downlink direction as the results are clearly visible when heavy traffic is going from 5G core to UE than the uplink traffic from the UE to the 5G core. We would like to provide additional clarification on the throughput shown in Fig.6.3. The downlink throughput for 1 connection from a single UE was around 40 Mbps because of the throughput limitation of several others components in the end-to-end system used for evaluation. As mentioned in the §6.3, we built our solution on top of OAIs 5G core, RAN and UE reference implementation.

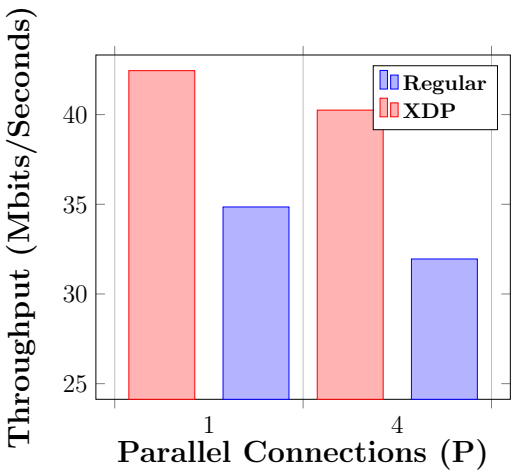


Figure 6.3: Performance Comparison Between Regular and XDP Socket - Downlink

6.5 PRAVEGA Design

In this section, we present the initial design of PRAVEGA. In this design, the GTP-U packets are completely handled in the kernel space without being sent to the user space.

6.5.1 Kernel Based CU-UP

Fig.6.4 illustrates the proposed design for the pure kernel based CU-UP. The Data Path Layer are configured by the Management Layer using eBPF Maps [127]. The detailed workflow of this design is shown in Fig.6.5. The key components of this design are discussed below:

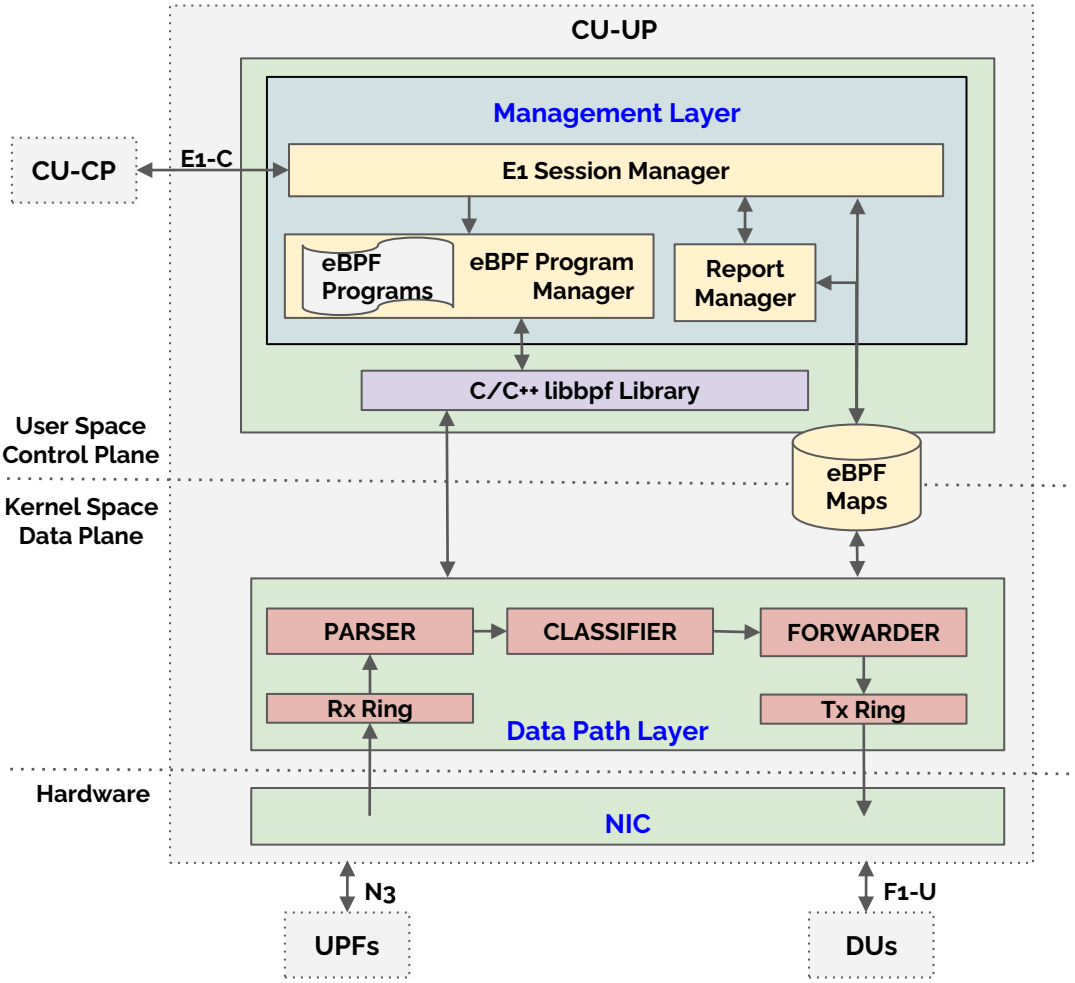


Figure 6.4: PRAVEGA- Pure Kernel eBPF Based CU-UP

Parser. The *'Parser'* component gets called for each packet received in the NIC. *'Parser'* filters and send GTP-U user plane packets to *'Classifier'* and pass the other type of packets to the network stack.

Classifier. The *'Classifier'* which processes the SDAP layer, retrieves the PDU session and QFI value from each packet. Next, it validates the PDU session information and assign the respective DRB for each packet based on the QFI-DRB Mapping. Finally, it forwards the packet to the *'Forwarder'*.

Forwarder. The *'Forwarder'* which processes the PDCP layer, consists of a set of DRBs per UE. Each packet passes through the respective DRB undergoing integrity protection, ciphering and sequence numbering. Finally, it routes the modified GTP-U packet to the respective DU based on the established PDU session.

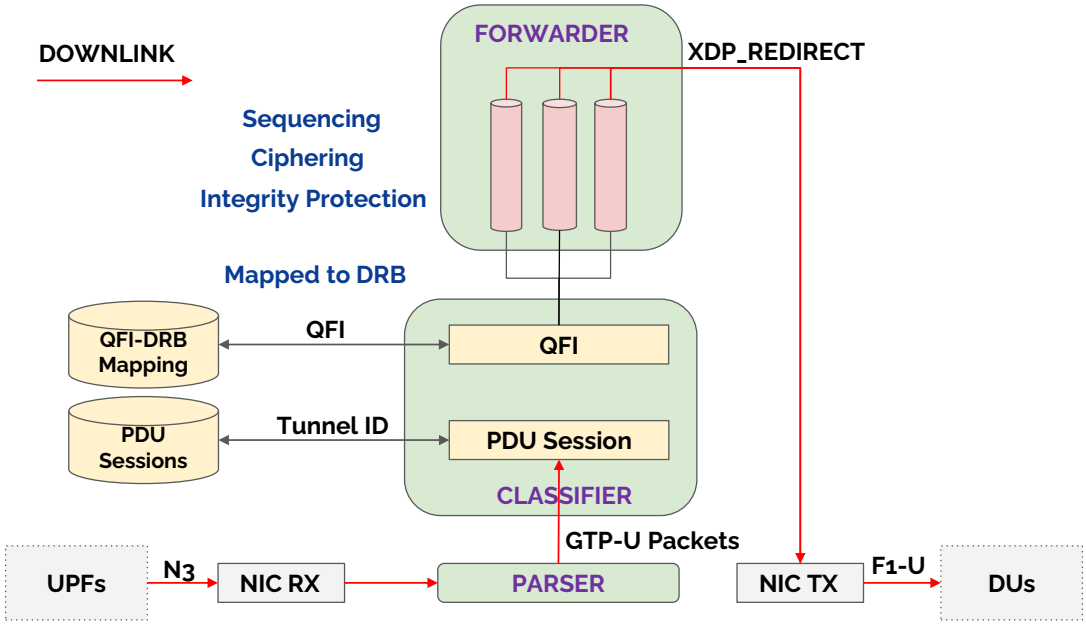


Figure 6.5: Pure Kernel eBPF Based CU-UP Flow - Downlink

6.6 Additional Design Options

To improve performance and better utilize computational resources, we explore dynamically offloading operations to the DPU, particularly *ciphering*. As it needs to perform operations on every bit of the packet payload, ciphering is CPU intensive [129] and can

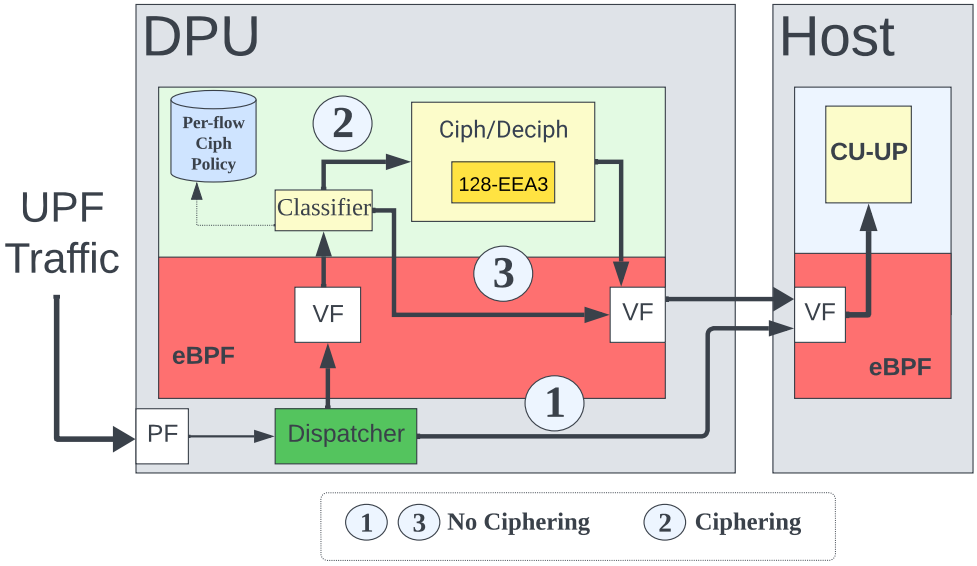


Figure 6.6: Ciphering Offloading Design

pollute the cache (which is crucial for the performance of stateful packet processing on the host [130, 131, 132], and mitigate interference on other cores [133]). To improve CPU utilization, cache efficiency, and leverage cost-efficiency cores on DPU, we design a ciphering operations offloading policy and dynamically offload the ciphering/deciphering operations to the SmartNIC. The dynamically offloading policy can be based on, but not limited to, the following factors: the high-level policy associated with the QoS flows on the types of traffic (whether it is sensitive information), the movement of the users (whether user move from a secure environment to an insecure environment), and the load current hardware infrastructure can handle.

Fig.6.6 illustrates the design of the offloading of ciphering operations of the downlink traffic path on 5G CU-UP. When traffic arrives at the DPU, the hardware can provide coarse-grained classification based on whether the traffic needs ciphering based on the installed rules (such as OVS rules on Bluefield SmartNIC). If it does not, the traffic will be sent to the host directly for further processing. And if not sure, the traffic will be processed by the eBPF/DPDK based ciphering/deciphering network functions running in the general-purpose processing unit (core) on the SmartNIC. The network

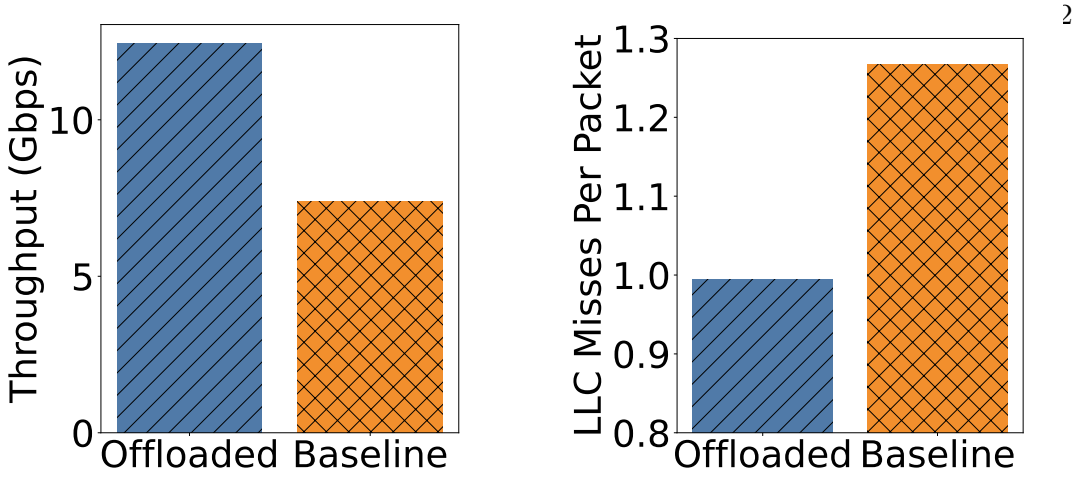


Figure 6.7: Preliminary Evaluation on Offloading Ciphering.

function consists of a stateful classifier that classifies packets based on the per-flow ciphering policy, which is dynamically changed based on the high-level policy and current environment. Based on the policy, the packet will be sent directly to the host or ciphered by the ciphering module. Using this design, we allow for a fine-grained policy to improve both the security and efficiency of the RAN system.

We have performed a preliminary evaluation (Fig.6.7) of the potential benefits on a server to show its effectiveness. We play a synthetic traffic of 1024-byte packets with 50% of the traffic that requires ciphering operations. With the offloading feature on, traffic is instead sent to the ciphering component in SmartNIC for ciphering as designed. From the result we show that offloading can improve the performance of the data plane and improve the utilization of the host cache.

6.7 Related Work

While eBPF and eBPF/XDP have been widely used in various (wired) networking and cloud computing systems, for example, to optimize service mesh/serverless computing [134], their application to 5G networks are rather limited. We are aware of only two recent papers, both apply eBPF/XDP to 5G core networks. In [135], eBPF/XDP based 5G UPF is implemented and they deploy it in a restrictive environment such as MEC (multi-access edge computing). In another paper [136], the focus is again on

eBPF/XDP kernel-based 5G UPF with fallback on user-space for more complex packet handling. In [137], the authors proposed a 5G Mobile Gateway based on eBPF/XDP, but the solution is completely implemented in the 5G core. None of these works consider applying eBPF/XDP to 5G RAN. In a two-page poster paper [138], we outlined an initial design of a purely kernel-based CU-UP design. Building on this, in this chapter we advance, implement and evaluate the combination of user-space+kernel-based CU-UP framework.

6.8 Summary

In this chapter, we designed, and evaluated PRANAVAM, an eBPF+XDP based framework for open, programmable and scalable NextG RANs. Our evaluation shows that there is more than 22% improvement in PRANAVAM when compared to using regular sockets. We also discuss the design of offloading the ciphering operations to the SmartNIC using fine-grained, dynamic per-flow ciphering policy to improve the efficiency of the host and the security of the 5G RAN. As a future work, we are planning to implement the entire data plane layers of each components in the NextG system using eBPF and evaluate our system using real-world workload.

Chapter 7

Conclusion

In this thesis, we proposed a few solutions to improve the interoperability in the IoT Ecosystem. Although we came up with an approach to support interoperability from the IoT gateway to the cloud, there are still challenges connecting multi-vendor downstream IoT devices to a vendor-specific IoT gateway, especially due to the vendor-specific security mechanisms implemented in the IoT gateway and IoT hub. The proposed enhancements to the message subscriptions proves the possibility of local configuration and control of the message subscriptions framework and also enhances the message subscriptions functionality provided by the respective CSPs by adding an enable or disable option to the individual message subscription without any configuration changes in the IoT hub. The evaluation shows that the proposed method didn't increase any significant latency in the message communication between the entities. Using our proposed simulation framework, we were able to simulate multi-vendor specific IoT devices in a single simulation framework. The régulateur disables the communication between any entities when the specific message topic or endpoint or payload matches the local configuration to disable the communication. As of now, it doesn't disable communication only between two entities. But in future, we can extend this functionality to disable communication only between two entities. The régulateur is implemented as a module but we have a plan to integrate this functionality to the IoT gateway's runtime itself to decrease the latency. And also there are no standard message subscriptions framework followed by the CSPs, AWS uses MQTT topic and Azure uses endpoints, we are planning to propose a standard message subscriptions framework in IoT gateway. We

didn't run any server instances in the IoT hub to collect any metrics to understand and analyze the performance impact in the IoT hub, so our future work will focus on extending the performance evaluation to the IoT hub.

We have presented *Kaala 2.0* – a modelling, simulation and emulation platform that are capable of creating IoT devices of various types. Using our proposed simulation framework, we were able to simulate multi-vendor specific IoT devices in a single simulation framework. We also simulated real-time events like fire in a room/building scenario and evaluated how this work can be extended for other real-time scenarios. We were able to simulate devices which can generate large amount of data to verify and validate 5G technology.

We argued for the need to shift the way we develop applications for 5G to utilize ML throughput prediction, adaptive content bursting, dynamic radio(band) switching to make video streaming applications 5G-aware. Using real-world 5G traces, our results show these mechanisms can improve user's QoE, despite wildly varying 5G throughput.

We designed, implemented, and evaluated HyperRAN, a fine-grained, cross-edged, QoS framework that provides an enhanced QoE for emerging applications. Our framework is first to support emulation of multiple channels simultaneously. HyperRAN also excels in performance when compared to other available open-source RAN systems. The Hyper Scheduler is integrated with the O-RAN RIC and send metrics. Our system leverages these features to provide significant stability and improved QoS and QoE for relevant use-cases in volumetric video and LiDAR streaming. In the future, we plan to employ machine learning algorithms on the collected data to more efficiently use the radio resources. We would also like to exploit the LiDAR data and perform packet-based QoS approach to efficiently prioritize critical data and to drop/de-prioritize irrelevant/lower priority frames. Finally, we hope to examine real-time smart prioritization of multi-modal data streams.

Finally, we designed, and evaluated PRANAVAM, an eBPF+XDP based framework for open, programmable and scalable NextG RANs. Our evaluation shows that there is more than 22% improvement in PRANAVAM when compared to using regular sockets. We also discuss the design of offloading the ciphering operations to the SmartNIC using fine-grained, dynamic per-flow ciphering policy to improve the efficiency of the host and the security of the 5G RAN. As a future work, we are planning to implement the entire

data plane layers of each components in the NextG system using eBPF and evaluate our system using real-world workload.

References

[1] Deloitte. Rewriting the rules for the digital age, 2017.

[2] 3GPP. 5G NR; Physical channels and modulation. Technical Specification (TS) 38.211, 3rd Generation Partnership Project (3GPP), 04 2018. Version 14.2.2.

[3] O-RAN Alliance. O-ran alliance, 2023.

[4] Sharu Bansal and Dilip Kumar. Iot ecosystem: A survey on devices, gateways, operating systems, middleware and communication. *International Journal of Wireless Information Networks*, pages 1–25, 2020.

[5] Jinhwan Jung, Jihoon Ryoo, Yung Yi, and Song Min Kim. Gateway over the air: towards pervasive internet connectivity for commodity iot. In Eyal de Lara, Iqbal Mohomed, Jason Nieh, and Elizabeth M. Belding, editors, *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 54–66. ACM, 2020.

[6] Aditya Nugur, Manisa Pipattanasomporn, Murat Kuzlu, and Saifur Rahman. Design and development of an iot gateway for smart building applications. *IEEE Internet Things J.*, 6(5):9020–9029, 2019.

[7] Partha Pratim Ray, Nishant Thapa, and Dinesh Dash. Implementation and performance analysis of interoperable and heterogeneous iot-edge gateway for pervasive wellness care. *IEEE Trans. Consumer Electron.*, 65(4):464–473, 2019.

[8] Tolulope Adesina and Oladiipo Osasona. A novel cognitive iot gateway framework: Towards a holistic approach to iot interoperability. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 53–58. IEEE, 2019.

[9] Jose Macias, Harold Pinilla, Wilder E. Castellanos, José Alvarado, and Andres Sánchez. Design and implementation of a multiprotocol iot gateway. *CoRR*, abs/2001.08171, 2020, 2001.08171.

[10] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, 24(3):796–809, 2019.

[11] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[12] O. Bello M. Chernyshev, Z. Baig and S. Zeadally. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2018.

[13] Blair Salama, Elkhatib. Iotnetsim: A modelling and simulation platform for end-to-end iot services and networking. *ACM SIGCOMM Computer Communication Review*, 2019.

[14] AWS. Iot device simulator.

[15] 3rd Generation Partnership Project. Release 15. `https://www.3gpp.org/release-15`, April 2020.

[16] 3rd Generation Partnership Project. Release 16. `https://www.3gpp.org/release-16`, July 2020.

[17] 3rd Generation Partnership Project. Release 17. `https://www.3gpp.org/release-17`, March 2021.

[18] RF Wireless World. 5QI table — 5G QoS identifier values and meanings. `https://www.rfwireless-world.com/5G/5G-QoS-Identifier-5QI.html`. Access: 03/17/2023.

[19] Devopedia. 5G Quality of Service. `https://devopedia.org/5g-quality-of-service`. Access: 03/17/2023.

[20] O-RAN ALLIANCE. O-RAN Use Cases and Deployment Scenarios. https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5e95a0a306c6ab2d1cbca4d3/1586864301196/O-RAN+Use+Cases+and+Deployment+Scenarios+Whitepaper+February+2020.pdf.

[21] Wikipedia. Gprs tunnelling protocol, 2022.

[22] Common Public Radio Interface. Specification overview, 2023.

[23] Altexsoft. Making sense of iot platforms: Aws vs azure vs google vs ibm vs cisco, 2020.

[24] Microsoft. Configure an iot edge device to act as a transparent gateway.

[25] Ieee standard glossary of software engineering terminology. *ANSI/ IEEE Std 729-1983*, pages 1–40, 1983.

[26] J. Kiljander, A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, J. Soininen, and T. S. Cinotti. Semantic interoperability architecture for pervasive computing and internet of things. *IEEE Access*, 2:856–873, 2014.

[27] ebpf - introduction, tutorials community resources, 2023.

[28] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 54–66, New York, NY, USA, 2018. Association for Computing Machinery.

[29] AWS IoT. Aws iot developer guide, 2020.

[30] Google Cloud . Overview of internet of things — solutions — google cloud, 02 2019.

[31] Google Cloud. Publishing over the http bridge — cloud iot core documentation, 2019.

[32] Google Cloud. Publishing over the mqtt bridge — cloud iot core documentation, 2019.

[33] Google IoT Core. Using json web tokens (jwts) — cloud iot core documentation, 06 2020.

[34] Amazon AWS. Aws lambda – product features, 2019.

[35] AWS. Configure devices and subscriptions.

[36] Microsoft. Deploy azure iot edge modules from the azure portal.

[37] Docker. What is a container?

[38] AWS. Run lambda functions on the aws iot greengrass core - aws iot greengrass, 11 2018.

[39] AWS. Machine learning inference with aws iot greengrass solution accelerator, 10 2019.

[40] Microsoft kgremban. Learn how the runtime manages devices - azure iot edge, 11 2019.

[41] Microsoft. Understand azure iot hub device twins, February 2020.

[42] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt).

[43] Google Cloud. Devices, configuration, and state — cloud iot core documentation, 06 2020.

[44] UNIFY-IoT Project. Deliverable d03.01, report on iot platform activities, 2016.

[45] Google. Using gateways.

[46] Udhaya Kumar Dayalan, Rostand AK Fezeu, Nitin Varyani, Timothy J Salo, and Zhi-Li Zhang. Eciot: Case for an edge-centric iot gateway. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 154–156, 2021.

[47] Microsoft Azure. Azure/azure-iot-sdks, 12 2020.

[48] CHIP. Project connected home over ip.

[49] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2017.

[50] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo, and Zhi-Li Zhang. Veeredge: Towards an edge-centric iot gateway. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 690–695, 2021.

[51] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Timothy J. Salo, and Zhi-Li Zhang. Kaala: Scalable, end-to-end, iot system simulator. In *Proceedings of the ACM SIGCOMM Workshop on Networked Sensing Systems for a Sustainable Society*, page 33–38, 2022.

[52] Mininet. Mininet - an instant virtual network on your laptop (or other pc).

[53] Docker. Get started with docker.

[54] IETF. Real time streaming protocol (rtsp).

[55] Open Air Interface. Open air interface, 2022.

[56] AWS. aws/aws-iot-device-sdk-embedded-c, 12 2020.

[57] Xuezhi Zeng, Saurabh Kumar Garg, Peter Strazdins, Prem Prakash Jayaraman, Dimitrios Georgakopoulos, and Rajiv Ranjan. Iotsim: A simulator for analysing iot applications. *Journal of Systems Architecture*, 72:93–107, 2017.

[58] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[59] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[60] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[61] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[62] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2008.

[63] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. Oversim: A flexible overlay network simulation framework. In *2007 IEEE global internet symposium*, pages 79–84. IEEE, 2007.

[64] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. Planetsim: A new overlay network simulation framework. In *International Workshop on Software Engineering and Middleware*, pages 123–136. Springer, 2004.

[65] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim, and Chita R Das. Mdcsim: A multi-tier data center simulation, platform. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–9. IEEE, 2009.

[66] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[67] Maria Salama, Yehia Elkhatib, and Gordon Blair. Iotnetsim: A modelling and simulation platform for end-to-end iot services and networking. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 251–261, 2019.

[68] Udhaya Kumar Dayalan, Timothy J. Salo, Rostand A. K. Fezeu, and Zhi-Li Zhang. Kaala 2.0: Scalable iot/nextg system simulator. *IEEE Network*, 37(3):240–246, 2023.

[69] Wei Ye, Jason Carpenter, Zejun Zhang, Rostand AK Fezeu, Feng Qian, and Zhi-Li Zhang. A closer look at stand-alone 5g deployments from the ue perspective. In *2023 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 86–91. IEEE, 2023.

[70] T-Mobile. Teleoperated driving: safely controlling cars remotely. `https://www.t-systems.com/de/en/industries/automotive/connected-mobility/teleoperated-driving`. Access: 03/17/2023.

[71] srsRAN Project. srsRAN, 2023.

[72] Eman Ramadan, Arvind Narayanan, Udhaya K. Dayalan, Rostand A. K. Fezeu, Feng Qian, and Zhi-Li Zhang. Case for 5g-aware video streaming applications. In *Proceedings of the ACM SIGCOMM Workshop on 5G Measurements, Modeling, and Use Cases*, 5G-MeMU'21, 2021.

[73] Jason Carpenter, Wei Ye, Feng Qian, and Zhi-Li Zhang. Multi-modal vehicle data delivery via commercial 5g mobile networks: An initial study. In *2023 IEEE 43rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 157–162. IEEE, 2023.

[74] Anis Elgabli, Vaneet Aggarwal, Shuai Hao, Feng Qian, and Subhabrata Sen. Lbp: Robust rate adaptation algorithm for svc video streaming. *IEEE/ACM Transactions on Networking*, 26(4):1633–1645, 2018.

[75] Yunzhuo Liu, Bo Jiang, Tian Guo, Ramesh K Sitaraman, Don Towsley, and Xinbing Wang. Grad: Learning for overhead-aware adaptive video streaming with scalable video coding. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 349–357, 2020.

[76] Siyuan Xiang, Min Xing, Lin Cai, and Jianping Pan. Dynamic rate adaptation for adaptive video streaming in wireless networks. *Signal Processing: Image Communication*, 39:305–315, 2015.

[77] Arvind Narayanan, Eman Ramadan, et al. Lumos5G: Mapping and Predicting Commercial MmWave 5G Throughput. In *ACM IMC'20*, 2020.

[78] Qualcomm. How 5G can transform telemedicine to tackle today's toughest challenges. `https://www.qualcomm.com/news/onq/2021/01/how-5g-can-transform-telemedicine-tackle-todays-toughest-challenges`. Access: 11/20/2023.

[79] Techwire Asia. 5G is key for AI autonomous warehouses. `https://techwireasia.com/2023/10/how-is-5g-enhancing-ai-autonomous-warehouses/`. Access: 11/20/2023.

[80] ATT. ATT Participates in Department of Defense Demonstration of 5G-enabled Smart Warehouse Solutions at Naval Base Coronado. `https://www.rcrwireless.com/20220609/5g/att-participates-dod-demo-5g-enabled-smart-warehouse-solutions`. Access: 11/20/2023.

[81] The Linux Foundation Projects. ONAP. `https://www.onap.org/`. Access: 03/17/2023.

[82] Raymond Kwan, Cyril Leung, and Jie Zhang. Proportional fair multiuser scheduling in lte. *IEEE Signal Processing Letters*, 16(6):461–464, 2009.

[83] Christian Wengerter, Jan Ohlhorst, and Alexander Golitschek Edler von Elbwart. Fairness and throughput analysis for generalized proportional fair frequency scheduling in ofdma. In *2005 IEEE 61st vehicular technology conference*, volume 3, pages 1903–1907. IEEE, 2005.

[84] Yongzhou Chen, Ruihao Yao, Haitham Hassanieh, and Radhika Mittal. Channel-Aware 5g RAN slicing with customizable schedulers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1767–1782, Boston, MA, April 2023. USENIX Association.

[85] Open5GS. Open5gs, 2023.

[86] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. Flexric: An sdk for next-generation sd-rans. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, page 411–425, New York, NY, USA, 2021. Association for Computing Machinery.

[87] Nvidia bluefield data processing units, 2023.

[88] Nvidia bluefield modes of operation, 2023.

[89] Data plane development kit, 2023.

[90] Nvidia doca dpi programming guide, 2023.

[91] Accuver. Accuver — XCAL. `http://accuver.com/sub/products/view.php?idx=6`, 2020.

[92] Paul Bender, Peter Black, Matthew Grob, Roberto Padovani, Nagabhushana Sindhushyana, and Andrew Viterbi. Cdma/hdr: a bandwidth efficient high speed wireless data service for nomadic users. *IEEE Communications magazine*, 38(7):70–77, 2000.

[93] A Jalali, R Padovani, and R Pankaj. Data throughput of cdma-hdr a high efficiency-high data rate personal communication wireless system. In *VTC2000-Spring. 2000 IEEE 51st Vehicular Technology Conference Proceedings (Cat. No. 00CH37026)*, volume 3, pages 1854–1858. IEEE, 2000.

[94] Harold J Kushner and Philip A Whiting. Asymptotic properties of proportional-fair sharing algorithms. Technical report, BROWN UNIV PROVIDENCE RI DIV OF APPLIED MATHEMATICS, 2002.

[95] Matthew Andrews and Lisa Zhang. Scheduling algorithms for multi-carrier wireless data systems. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 3–14, 2007.

[96] David N. C. Tse, Pramod Viswanath, and Lizhong Zheng. Diversity-multiplexing tradeoff in multiple-access channels. *IEEE Transactions on Information Theory*, 50(9):1859–1874, 2004.

[97] J Nicholas Laneman, David NC Tse, and Gregory W Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Transactions on Information theory*, 50(12):3062–3080, 2004.

[98] Sem Borst. User-level performance of channel-aware scheduling algorithms in wireless data networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 1, pages 321–331. IEEE, 2003.

[99] Krister Norlund, Tony Ottosson, and Anna Brunstrom. Fairness measures for best effort traffic in wireless networks. In *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No. 04TH8754)*, volume 4, pages 2953–2957. IEEE, 2004.

[100] Matthew Andrews, Krishnan Kumaran, Kavita Ramanan, Alexander Stolyar, Phil Whiting, and Rajiv Vijayakumar. Providing quality of service over a shared wireless link. *IEEE Communications magazine*, 39(2):150–154, 2001.

[101] Matthew Andrews. A survey of scheduling theory in wireless data networks. *Wireless Communications*, pages 1–17, 2007.

[102] Cédric Westphal. Monitoring proportional fairness in cdma2000r.

[103] Akhilesh Pokhariyal, Guillaume Monghal, Klaus I Pedersen, Preben E Mogensen, Istvan Z Kovacs, Claudio Rosa, and Troels E Kolding. Frequency domain packet scheduling under fractional load for the utran lte downlink. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 699–703. IEEE, 2007.

[104] Akhilesh Pokhariyal, Klaus I Pedersen, Guillaume Monghal, Istvan Z Kovacs, Claudio Rosa, Troels E Kolding, and Preben E Mogensen. Harq aware frequency domain packet scheduler with different degrees of fairness for the utran long term evolution. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 2761–2765. IEEE, 2007.

[105] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as*

part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13), pages 459–471, 2013.

[106] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 509–522, 2015.

[107] Feng Lu, Hao Du, Ankur Jain, Geoffrey M Voelker, Alex C Snoeren, and Andreas Terzis. CQIC: Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2015.

[108] Thomas Nitsche, Carlos Cordeiro, Adriana B Flores, Edward W Knightly, Eldad Perahia, and Joerg C Widmer. Ieee 802.11 ad: directional 60 ghz communication for multi-gigabit-per-second wi-fi. *IEEE Communications Magazine*, 52(12):132–141, 2014.

[109] Sanjib Sur, Ioannis Pefkianakis, Xinyu Zhang, and Kyu-Han Kim. Wifi-assisted 60 ghz wireless networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 28–41. ACM, 2017.

[110] Sanjib Sur, Xinyu Zhang, Parmesh Ramanathan, and Ranveer Chandra. Beamspy: enabling robust 60 ghz links under blockage. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 193–206, 2016.

[111] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating transparent web proxies in cellular networks. In *International Conference on Passive and Active Network Measurement*, pages 262–276. Springer, 2015.

[112] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data*

Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, page 479–494. ACM, 2020.

[113] Qiang Ye, Junling Li, Kaige Qu, Weihua Zhuang, Xuemin Sherman Shen, and Xu Li. End-to-end quality of service in 5g networks: Examining the effectiveness of a network slicing framework. *IEEE Vehicular Technology Magazine*, 13(2):65–74, 2018.

[114] Robert Margolies, Ashwin Sridharan, et al. Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms. *IEEE/ACM Transactions on Networking (TON)*, 24(1):355–367, 2016.

[115] A. Chakraborty, M. S. Rahman, H. Gupta, and S. R. Das. Specsense: Crowdsensing for efficient querying of spectrum occupancy. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[116] Emmanouil Alimpertis, Athina Markopoulou, Carter Butts, and Konstantinos Psounis. City-wide signal strength maps: Prediction with random forests. In *The World Wide Web Conference*, WWW '19, page 2536–2542, New York, NY, USA, 2019. Association for Computing Machinery.

[117] A. Samba, Y. Busnel, A. Blanc, P. Dooze, and G. Simon. Instantaneous throughput prediction in cellular networks: Which information is needed? In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management*, pages 624–627, 2017.

[118] Lifan Mei, Runchen Hu, Houwei Cao, Yong Liu, Zifa Han, Feng Li, and Jin Li. Realtime mobile bandwidth prediction using lstm neural network. In *International Conference on Passive and Active Network Measurement*, pages 34–47. Springer, 2019.

[119] Hongji Huang, Song Guo, Guan Gui, Zhen Yang, Jianhua Zhang, Hikmet Sari, and Fumiyuki Adachi. Deep learning for physical-layer 5g wireless techniques: Opportunities, challenges and solutions. *IEEE Wirel. Commun.*, 27(1):214–222, 2020.

[120] Zhijin Qin, Hao Ye, Geoffrey Ye Li, and Biing-Hwang Fred Juang. Deep learning in physical layer communications. *IEEE Wirel. Commun.*, 26(2):93–99, 2019.

[121] Alessio Zappone, Marco Di Renzo, and Mérouane Debbah. Wireless networks design in the era of deep learning: Model-based, ai-based, or both? *IEEE Trans. Commun.*, 67(10):7331–7376, 2019.

[122] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Artificial neural networks-based machine learning for wireless networks: A tutorial. *IEEE Commun. Surv. Tutorials*, 21(4):3039–3071, 2019.

[123] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Commun. Surv. Tutorials*, 21(3):2224–2287, 2019.

[124] Zhi-Li Zhang, Udhaya K. Dayalan, Eman Ramadan, and Timothy J. Salo. Towards a software-defined, fine-grained qos framework for 5g and beyond networks. In *Proceedings of the ACM SIGCOMM Workshop on Network Meets AI & ML*, NetAI'21, 2021.

[125] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Timothy J. Salo, and Zhi-Li Zhang. Prototyping a fine-grained qos framework for 5g and nextg networks using powder. In *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 416–419, 2022.

[126] Cisco. Cisco private 5g solution overview, 2023.

[127] Prototype Kernel. ebpf maps, 2023.

[128] iPerf3. iperf3. `https://iperf.fr/`, June 2023.

[129] Roberto Avanzi and Billy Bob Brumley. Faster 128-eea3 and 128-eia3 software. In *Information Security: 16th International Conference, ISC 2013, Dallas, Texas, November 13-15, 2013, Proceedings*, pages 199–208. Springer, 2015.

[130] Peng Zheng, Wendi Feng, Arvind Narayanan, and Zhi-Li Zhang. Nfv performance profiling on multi-core servers. In *2020 IFIP Networking Conference (Networking)*, pages 91–99. IEEE, 2020.

[131] Ziyan Wu, Tianming Cui, Arvind Narayanan, Yang Zhang, Kangjie Lu, Antonia Zhai, and Zhi-Li Zhang. Granularnf: Granular decomposition of stateful nfv at 100 gbps line speed and beyond. *ACM SIGMETRICS Performance Evaluation Review*, 50(2):46–51, 2022.

[132] Hamid Ghasemirahni, Tom Barbette, Georgios P Katsikas, Alireza Farshin, Amir Roozbeh, Massimo Girondi, Marco Chiesa, Gerald Q Maguire Jr, and Dejan Kostić. Packet order matters! improving application performance by deliberately delaying packets. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 807–827, 2022.

[133] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. {ResQ}: Enabling {SLOs} in network function virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 283–297, 2018.

[134] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. Spright: Extracting the server from serverless computing! high-performance ebpf-based event-driven, shared-memory processing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 780–794, New York, NY, USA, 2022. Association for Computing Machinery.

[135] Thiago A. Navarro do Amaral, Raphael V. Rosa, David F. Cruz Moura, and Christian E. Rothenberg. An in-kernel solution based on xdp for 5g upf: Design, prototype and performance evaluation. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 146–152, 2021.

[136] Christian Scheich, Marius Corici, Hauke Buhr, and Thomas Magedanz. Express data path extensions for high-capacity 5g user plane functions. In *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, eBPF '23, page 86–88, New York, NY, USA, 2023. Association for Computing Machinery.

[137] Federico Parola, Sebastiano Miano, and Fulvio Risso. A proof-of-concept 5g mobile gateway with ebpf. In *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*, SIGCOMM '20, page 68–69, New York, NY, USA, 2021. Association for Computing Machinery.

[138] Udhaya Kumar Dayalan, Ziyan Wu, Gaurav Gautam, Feng Tian, and Zhi-Li Zhang. Pravega: Scaling private 5g ran via ebpf/xdp. In *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, eBPF '23, page 89–91, New York, NY, USA, 2023. Association for Computing Machinery.

[139] Cisco. Five components of iot edge devices.

Appendix A

Publications

In addition to this dissertation, the presented work and results are also documented in the following published papers.

A.1 Publications by Date

- **Udhaya Kumar Dayalan**, Jason Carpenter, Ngan Nguyen, Wei Ye, Ziyan Wu and Zhi-Li Zhang. 2024. "HyperRAN: Towards a Fine-Grained, Semantics-Aware, Intelligent NextG Radio Access Network Architecture". Under Submission, 2024.
- **Udhaya Kumar Dayalan**, Ziyan Wu, Gaurav Gautam, Feng Tian, and Zhi-Li Zhang. 2023. "Towards an eBPF+XDP based Framework for Open, Programmable and Scalable NextG RANs". In 2023 IEEE Future Networks World Forum (FNWF '23).
- **Udhaya Kumar Dayalan**, Ziyan Wu, Gaurav Gautam, Feng Tian, and Zhi-Li Zhang. 2023. "PRAVEGA: Scaling Private 5G RAN via eBPF/XDP". In Proceedings of the ACM SIGCOMM 2023 1st Workshop on eBPF and Kernel Extensions (eBPF '23).
- **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Timothy J. Salo, and Zhi-Li Zhang. 2022. "Kaala 2.0: Scalable IoT/NextG System Simulator". In IEEE Network, vol. 37, no. 3, May/June 2023.

- **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Timothy J. Salo and Zhi-Li Zhang. 2022. "Prototyping a Fine-Grained QoS Framework for 5G and NextG Networks using POWDER". 2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS '22).

- **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Timothy J. Salo, and Zhi-Li Zhang. 2022. "Kaala: scalable, end-to-end, IoT system simulator". In Proceedings of the ACM SIGCOMM Workshop on Networked Sensing Systems for a Sustainable Society (NET4us '22).

- Zhi-Li Zhang, **Udhaya Kumar Dayalan**, Eman Ramadan, and Timothy J. Salo. 2021. "Towards a Software-Defined, Fine-Grained QoS Framework for 5G and Beyond Networks". In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI '21).

- Eman Ramadan, Arvind Narayanan, **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Feng Qian, and Zhi-Li Zhang. 2021. "Case for 5G-aware video streaming applications". In Proceedings of the 1st Workshop on 5G Measurements, Modeling, and Use Cases (5G-MeMU '21).

- **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo and Zhi-Li Zhang. 2021. "VeerEdge: Towards an Edge-Centric IoT Gateway". 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid '21).

- **Udhaya Kumar Dayalan**, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo, and Zhi-Li Zhang. 2021. "ECIoT: Case for an Edge-Centric IoT Gateway". In Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21).

Appendix B

Common IoT Terms

Below are some of the common IoT terms:

B.1 IoT device

IoT devices cannot directly connect to the internet [23]. These devices communicate over Zigbee or Bluetooth [45]. Some of the IoT device examples are motion sensor, light switch, water sensor, door/window sensor.

B.2 IoT edge

IoT edges have the ability to communicate to the IoT hub on the internet. Some of its common functionalities [139] include publishing telemetry events, getting configuration data, custom applications and providing offline support. IoT edge devices also store and process the data locally. They are capable of running machine learning models.These are stand-alone devices and doesn't not support downstream IoT devices as shown in Fig. 2.7.

B.3 IoT gateway

IoT gateway provides all the capability of an IoT edge and much more [24]. They act as a router connecting to the internet when the downstream IoT device can't directly

connect to the IoT hub, such as a ZigBee or Bluetooth device [45]. And also helps authenticating to IoT hub when the device can't send its own credentials, or to provide an additional layer of security by using the credentials of both the IoT device and the IoT gateway. IoT gateway also helps in translating protocols. For example, the IoT device communicates BACnet or Modbus protocol to the gateway and the gateway device translates it to MQTT protocol before sending it to the IoT hub.

B.4 IoT Hub

The IoT Hub is called using several other names like IoT Cloud, IoT core, etc. Basically this is the cloud side of the IoT architecture. The IoT hub connects, processes, stores, and analyzes data. IoT hub scales to multiple edge devices and have various capabilities based on the respective CSPs.

B.5 Modules

Modules run as containers [37] in the IoT gateway which is managed by the IoT gateway's run-time. Container is a unit of software that contains code and all its dependencies (run-time, system tools, system libraries and settings) as a single package. Containers [37] runs in most of the Operating System with the support of an engine. In Azure IoT, the containers package with custom code are called modules. And in AWS IoT, these containers are called as lambda functions [34]. Additionally, in AWS, the lambda functions can run as an individual process in the IoT gateway instead of a container. As shown in Figure 2.8, local database, web-server, machine learning services are some of the examples for a module.

B.6 Message Subscriptions

Message subscriptions is a concept in IoT which allows communication between various entities listed above in this section [35]. Because, by default these entities cannot communicate to each other. Message subscriptions is a term introduced by AWS IoT. In Azure IoT, the message subscriptions are called as routes [36]. For consistency, in

this thesis, we will be using message subscriptions while describing the functionality of Azure IoT as well. The message subscriptions are configured based on the MQTT message topic or endpoints. There are 3 fields required for a message subscription [35]. First, the 'source', from where the message originated. Next, the 'destination', to which the message needs to be sent. And finally, AWS requires a message 'topic' or Azure requires the 'endpoint'.

B.7 MQTT Broker

The MQTT broker is a key component in an IoT gateway. The downstream IoT devices talk to the gateway through the broker. Each vendor has specific security mechanisms added to authenticate downstream IoT devices which is discussed in detail in section 3.2.1. The MQTT broker receives the MQTT messages sent to them and forwards the MQTT messages based on the message subscriptions. If there are no configurations for a particular topic or endpoint based on the vendor-specific IoT gateway, then the MQTT message is dropped by the MQTT broker.