LUND UNIVERSITY

Taming Cloud Integrated Systems in the Wild

Peng, Haorui

2023

Link to publication

Citation for published version (APA):
Peng, H. (2023). *Taming Cloud Integrated Systems in the Wild*. Lund University.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Taming Cloud Integrated Systems in the Wild

Taming Cloud Integrated Systems in the Wild

by Haorui Peng

LUND
UNIVERSITY

Cover generated by Midjourney, edited by Haorui Peng.

Funding information: The thesis work was financially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation, the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), the IMMINENCE project funded by Sweden's Innovation Agency (VINNOVA), and the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF).

© Haorui Peng 2023

Department of Electrical and Information Technology
Faculty of Engineering, Lund University
Box 118
SE–221 00 LUND
Sweden

Series of licentiate and doctoral theses
No. 165
ISBN 978-91-8039-873-2 (print)
ISBN 978-91-8039-872-5 (pdf)
ISSN 1654-790X-165

Printed in Sweden by E-husets tryckeri, Lund University, Lund 2023

Dedicated to everyone I love and everyone who loves me.

Contents

List of publications

This thesis is based on the following publications:

i **5G Radio Access Network Slicing in Massive MIMO Systems for Industrial Applications**
Haorui Peng, Emma Fitzgerald, William Tärneberg, Maria Kihl
2020 Seventh International Conference on Software Defined Systems

ii **Is Cloud RAN a Feasible Option for Industrial Communication Network?**
Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
Journal of Communications Software and Systems, 2021, 17(2), pp. 97-106

iii **Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems**
Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
2023 International Conference on Software, Telecommunications and Computer Networks

iv **Evaluation of Control over the Edge of a Configurable Mid-band 5G Base Station**
Haorui Peng, William Tärneberg, Emma Fitzgerald, Fredrik Tufvesson, Maria Kihl
2022 IEEE 6th International Conference on Fog and Edge Computing

v **Latency-aware Radio Resource Allocation over Cloud RAN for Industry 4.0**
Haorui Peng, William Tärneberg, Maria Kihl
2021 International Conference on Computer Communications and Networks

vi **Punctual Cloud: Unbinding Real-time Applications from Cloud-induced Delays**
Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
2021 International Symposium on Networks, Computers and Communications

vii **Punctual Cloud: Achieving Punctuality for Time-Critical Cloud Control Systems**
Haorui Peng, Fatemeh Akbarian, William Tärneberg, Maria Kihl
2023 IEEE International Conference on Cloud Networking

Furthermore, I have contributed to the following papers which are not part of the thesis:

i **Latency prediction in 5G for control with deadtime compensation**

Johan Ruuskanen, **Haorui Peng**, Alexandre Martins
Proceedings of the Workshop on Fog Computing and the IoT, 2019, pp. 51-55

ii **Massive MIMO pilot scheduling over cloud ran for industry 4.0**

Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
2020 International Conference on Software, Telecommunications and Computer Networks

iii **FedApp: a Research Sandbox for Application Orchestration in Federated Clouds using OpenStack**
Johan Ruuskanen, **Haorui Peng**, Alfred Åkesson, Lars Larsson, Maria Kihl
arXiv preprint arXiv:2109.01480

iv **Massive MIMO pilot scheduling over Cloud RAN**

Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
16th Swedish National Computer Networking Workshop

Abstract

This thesis unfolds a journey into the realm of cloud integrated systems. More specifically, it explores the transformational role of diverse cloud infrastructure, be it public or private, centralized or edge-based, when integrated into traditional systems. In this transformation, the cloud assumes the vital role of controllers. Inevitably, this shift towards cloud integration also brings into play the expansive network that acts as the connective tissue between traditional systems and the cloud, adding another layer of complexity to the newly formed integrated system.

In this work, we shed light on the less-talked-about side of cloud integration. Beyond the evident benefits of this transition, we face an array of challenges that emerge along with the introduction of the cloud and its accompanying network. Adapting traditional system deployment to this new era of cloud-based computing is one such necessity. The advent of virtualization and container technologies introduces additional requirements for software management. Shared infrastructure mandates stricter control over incoming traffic. Furthermore, real-world networks often act unpredictably, straying from their simulated behaviours. Even the much-touted 5G technology has not completely lived up to the expectations set a decade ago.

However, the ambition of this thesis does not lie in the enhancement of existing infrastructure, the improvement of cloud technologies, or the acceleration of network speed. Rather, it aims to accept and work within the limitations and flaws inherent in both cloud and network infrastructures. The primary goal is to recognize the challenges these systems introduce, embrace their imperfections, and adapt our systems to work effectively with the realities of our imperfect cloud and unpredictable network environments.

To accomplish this, the thesis undertake a comprehensive analysis of two types of cloud integrated systems—Cloud RAN and Cloud Control System. A central focus is the evaluation of the practicality of implementing these systems using existing infrastructure. This evaluation is based on rigorous simulation as well as hands-on testbed experiments. In response to the insights gained from these assessments, the thesis proposes an innovative framework, built on a microservice architecture, to deploy cloud services more effectively for these systems. This framework is designed to mitigate the network latency impact brought on by unpredictable, "wild" environments. It does so by incorporating specialized prediction and estimation services, thereby enhancing the adaptability of these systems to real-world challenges.

Acknowledgements

Going through a PhD journey that has spanned over five years has been far from straightforward; it's been a demanding and, at times, exhausting experience. However, I am profoundly grateful for the incredible support around me that has made this challenging yet enriching journey possible.

First and foremost, I owe a debt of gratitude to my primary supervisor, Maria Kihl. I consider myself very lucky to have had her guidance throughout my PhD. Maria's unwavering support and mentorship have been invaluable. She provided me with a wealth of opportunities to participate in various fun projects, encouraging me to meet and talk with experts in the field. This experience significantly broadened my professional horizons and helped me find my own research interests. During one of the most challenging periods of my life and career, Maria stood beside me, taking proactive steps to ensure my well-being. Her steadfast support has been the bedrock upon which I have built my PhD journey.

My co-supervisor, William Tärneberg, although officially joining my advisory team in my second year, has been a great help to me from day one at office. William's passion for research is contagious, and he is full of endless creative ideas. Yet, what stands out the most is his empathetic nature and consistent willingness to assist me through any challenges. Whenever I have felt stuck or lost, William has been my go-to person for insightful and constructive feedback. His guidance has made the path to completing my PhD a far more manageable track.

Emma Fitzgerald, my other co-supervisor, is basically my role model for what an exceptional female researcher should be—intelligent, fully committed, and a natural at teamwork. Every discussion with her leaves me feeling inspired and methodically directed. Emma's enthusiasm for academic research is the kind of passion I wish to carry into my own career.

In addition to my supervisors, I would like to extend my heartfelt thanks to a number of senior colleagues in my department. Ulf Körner has always been the kindest figure in the office; his course was a highlight of my academic experience. Björn Landfeldt, with whom I have shared many lunchtime conversations, never fails to make our chats engaging and enjoyable. Stefan Höst, my office neighbour during the early years of my PhD, has been consistently supportive, always willing to help me solve big and small matters. A special nod to our administrative team—Elisabeth Nordström, Elisabeth Ohlsson, Linda Bienen, Erik Göthe and Sirvan Abdollah Poor, —for their patience and willingness to assist whenever I found myself mired in bureaucratic complexities.

Special mentions are due for Erik Jonsson, Lars Larsson and Fredrik Brumme. Erik,

our multitasking research engineer and network manager, has been endlessly patient, helping me with everything from cables issues and network firewalls to student lab management and even physically relocating heavy lab equipment. His assistance has been invaluable to my work and academic life. Lars, who visited our research group in my early PhD years, is a cloud computing expert whose enthusiasm inspired me to delve deeply into that area. Fredrik was instrumental in the installation of the 5G radio base station in our lab; his expertise broadened my understanding of mobile networks and set a high bar for what it means to be a senior engineer.

A Million thanks go out to my fellow PhD candidates in the department, who have been my reliable friends throughout this lengthy journey. Xuhong has been a trusted listener for many of my troubles at work. Hassan and Farnaz, who have since graduated and moved on, provided invaluable guidance and support during my initial days in the department. Zahra and Fatemeh are perhaps the ones with whom I have shared the most work hours, and I'm genuinely thankful for their camaraderie and enjoyed all the long and short trips we had together. Sahar became my office neighbour in my final year, and our shared music tastes have been a refreshing discovery—I look forward to many more concerts together. A special shoutout goes to Azra, Rikard, Sidra, Ilayda, Suleyman, Hedieh, Ziyu, Baichuan and anyone I may have unintentionally left out. Our banter, shared complaints, and jokes about the PhD life have made this path infinitely more bearable. Your encouragement and emotional support have been essential in helping me cross the finish line. Thank you all.

I cannot overlook the unwavering support of my friends, both in Sweden and China, who have been steadfast companions throughout these formative years. Di Wang was not only the first person I met upon arriving in Sweden but also my first and last flatmate here; her friendship and constant encouragement have been incredibly meaningful to me. Bingran and Ran Duan have been like a second family, always at the ready to help me navigate through any challenges that came my way since our paths first crossed. Special thanks to Xiaojie, Xin Zhang and Zhiqi for the laughter-filled weekends and memorable game and movie nights that lightened my spirit. And I'm forever grateful to Zehui, Fang Huang and Haoran for the wonderful homemade meals and delightful dinner evenings that we have shared.

Last but not least, my deepest, most heartfelt gratitude is reserved for my cherished family. My fur babies, Lina and Punchy, have been constant sources of comfort and joy. Their presence has become so integral to my life that I can't imagine my days without them. I hope we share many more years of happiness together.

As for my parents, my grandmother, and my dearly departed grandfather, their unwavering support has shaped me into the person I am today—a person I'm proud to be. Even though I've been away from home for years, their concern for my well-being

never wanes, reminding me of their endless love. They have supported every decision I have made, without question, granting me the freedom to be myself. I'm extraordinarily fortunate to belong to such an understanding and supportive family. My love for you will remain steadfast throughout my life.

爸爸妈妈, 外公外婆, 我永远爱你们!

Haorui Peng
Lund, October 2023

Acronyms

3GPP 3rd Generation Partnership Project

5G Fifth Generation Wireless Specifications

AI Artificial Intelligence

AWS Amazon Web Services

BBU BaseBand processing Unit

BnB Ball-and-Beam

CAN Controller Area Network

CCS Cloud Control System

CDN Content Delivery Network

CNI Container Network Interface

CSI Channel State Information

CN Core Network

CU Critical Unit

DRX Discontinuous Reception

DSP Digital Signal Processor

ECS Elastic Container Service

eMBB Enhanced Mobile Broadband

EDF Earliest Deadline First

EMA Exponential Moving Average

EPC Evolved Packet Core

ERDC Ericsson Research Data Center

ETSI European Telecommunications Standards Institute

FCFS First Come First Served

FDD Frequency-Division Duplexing

GCP Google Cloud Platform

GPP General-purpose Processing

HOL Head-Of-Line

IaaS Infrastructure-as-a-Service

ICMP Internet Control Message Protocol

IAE Integral Absolute Error

IoT Internet of Things

IIoT Industrial Internet of Things

ISP Internet Service Provider

LAN Local Area Network

LTE Long Term Evolution

LQR Linear Quadratic Regulator

LuMaMi Lund Massive MIMO

MAC Medium Access Control

MEC Mobile Edge Computing

MIMO multiple-input multiple-output

mMTC Massive Machine Type Communication

MNO Mobile Network Operator

MPC Model Predictive Control

MQ Message Queue

MQTT Message Queue Telemetry Transport

NIST National Institute of Standards and Technology

non-CU non-Critical Unit

NR New Radio

NSA Non Stand-Alone

OFDM Orthogonal Frequency-Division Multiplexing

PaaS Platform-as-a-Service

PDF Probability Density Function

PHY Physical Layer

QoS Quality of Service

RAN Radio Access Network

RBS Radio Base Station

PTO Probe Timeout

RTO Retransmission Timeout

RRH Remote Radio Head

RRQ Round Robin with partial Queuing information

RRNQ Round Robin with No Queuing information

RTT Round Trip Time

SA Stand-Alone

SaaS Software-as-a-Service

SDN Software Defined Networks

SDR Software Defined Radio

TDD Time Division Duplex

UE User Equipment

URLLC Ultra-Reliable and Low-Latency Communication

VM Virtual Machine

VNF Virtual Network Function

VXLAN Virtual Extensible LAN

WAN Wide Area Network

OAM Operations, Administration and Management

Taming Cloud Integrated Systems in the Wild

Part I

Introduction

Chapter 1

Cloud Integrated Systems

The process of cloud transformation has emerged as a revolutionary trend in recent years, radically altering the landscape of system software deployment. Traditionally, complex computational tasks required substantial hardware support, often leading to increased costs and complex management procedures. However, the advent of cloud computing has provided an efficient solution to this predicament. By offloading these computation-intensive tasks to the cloud, system deployment and maintenance become both more economical and simplified compared to traditional models.

This thesis delves into the exploration and evaluation of two pivotal cloud integrated systems: the Cloud Radio Access Networks (RANs) and Cloud Control Systems (CCSs). These systems epitomize the paradigm shift from conventional mechanisms to innovative cloud integrated frameworks. Central to both is the "control over the cloud" approach, characterized by a symbiotic relationship between the control object and its cloud controller.

The Cloud RAN stands out as a transformative model that reimagines traditional Radio Base Station (RBS) deployments. By relocating the baseband processing functions to the cloud, it infuses operations with greater efficiency, flexibility, and scalability. Within this context, the thesis spotlights the Medium Access Control (MAC) over the Cloud, emphasizing real-time resource allocation for diverse network subscribers. The instantaneous response from the cloud controller to resource requests takes centre stage here.

Cloud Control Systems (CCSs), on the other hand, represent a significant shift in the operation of traditional cyber-physical systems. These systems encompass tangible control entities, ranging from robots to power grids and intricate industrial processes. Characteristically, a CCS is defined by a feedback loop interlinking the control ob-

ject and the cloud controller. The overarching objective is to maintain stability and robustness in the control object, aligning it to a reference point. Transitioning the system controllers to the cloud propels these systems towards a centralized, efficient control structure.

Such cloud integrated systems, also referred to as systems "controlled over the cloud", are representative of a new wave of cloud-dependent operations. While such control harnesses the expansive benefits of cloud integration, it also confronts distinct challenges, particularly due to its dependence on consumer processes and the intricate cloud network and computing infrastructures. With their mission-critical nature, systems under "control over the cloud" necessitate research that addresses real-world conditions, foregrounding both scalability and consistent low latency amidst high-demand, real-time tasks.

Furthermore, the unpredictability of the network that bridges the cloud and other system components also poses a primary challenge. As control responsibilities transition to the cloud, this network becomes more than just a connector and often turns into an unpredictable variable in the system. Consequently, these systems are not just managed through the cloud but also traverse these erratic networks, adding a layer of complexity that this thesis delves into and aims to resolve.

This thesis primarily seeks to understand the challenges posed by cloud integrated systems and to adapt these systems to the "wild" environment without altering that environment. This remaining of this chapter offers an overview of the two cloud integrated systems addressed by this thesis: the Cloud RAN and Cloud Control Systems (CCSs). Chapter 2 subsequently delves into the dual facets constituting the "wild", namely cloud computing and networking, both of which introduce complexities for cloud integrated systems. Delving deeper, Chapter 3 elaborates on the complexities of deploying these cloud integrated systems, with particular emphasis on their integration within industrial contexts – the primary environment this research targets for system adaptation and deployment. Lastly, Chapter 4 provides an overview of the technical content covered in this thesis, detailing the author's contributions to each of the underlying papers upon which the thesis is founded.

1.1 Cloud RAN

Cloud RAN is an advanced RAN architecture designed for the Fifth Generation Wireless Specifications (5G) and the beyond, emphasizing software-defined operations and centralized resource management in radio access networks. It's positioned as a prospective architecture for future RBSs, wherein signal processing functions are

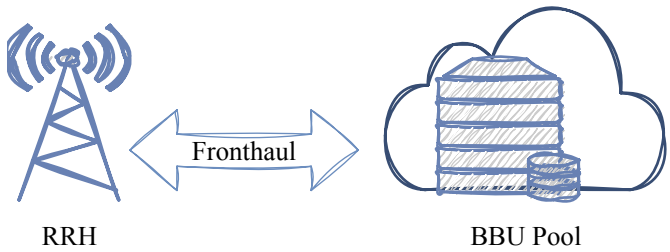


Figure 1.1: A Cloud RAN system.

either partially or fully centralized in a BaseBand processing Unit (BBU) pool, leveraging virtualization and cloud capabilities. Its fundamental goal is to offer efficient and cost-effective mobile Internet access.

This innovation can be seen as a progression from C-RAN, which stands for Centralized processing, Cooperative radio, Cloud, and Green infrastructure Radio Access Network [Chi11]. At its core, Cloud RAN seeks to harness centralized and collaborative baseband processing on real-time Cloud infrastructure, ideally leveraging General-purpose Processing (GPP) resources to minimize costs and associated risks [Gua+12].

In practical terms, Cloud RAN reconfigures the traditional setup. As the example depicted in Figure 1.1, it employs a splitting architecture by segregating the BBU from numerous conventional RBSs and centralizes them into a singular BBU pool within a cloud infrastructure. The subsequent Remote Radio Heads (RRHs) are minimized to handle basic radio-frequency operations such as transmission, reception, and analogue to digital conversion. Through this centralized BBU pool, cooperative baseband signal processing across multiple RRH sites becomes feasible. The BBU pool is linked with the RRH through fronthaul links. While optic cables, same as those used in traditional RBSs, are typically employed as the fronthaul links, there's an increasing interest in exploring more cost-effective alternatives. Notably, recent studies have been delving into the potential of employing Ethernet as the fronthaul link for Cloud RAN, showcasing its viability and benefits [Gom+18; Ass+17; Mou+17].

Yet, the full integration and software adaptation into Cloud RAN systems, as envisioned by [Chi11], necessitates a phased approach. The initial step to shape Cloud RAN merely demands the separation of BBUs from various traditional RBSs, centralizing them for baseband processing. However, this centralized BBU remains neither pooled nor virtualized. Ultimately, the Cloud RAN vision encompasses a fully pooled computational resource base in the form of a BBU pool. By this stage, Cloud RAN will not only embrace the merits of Software Defined Radio (SDR) but also incorporate virtualization technologies, favouring GPPs over Digital Signal Proces-

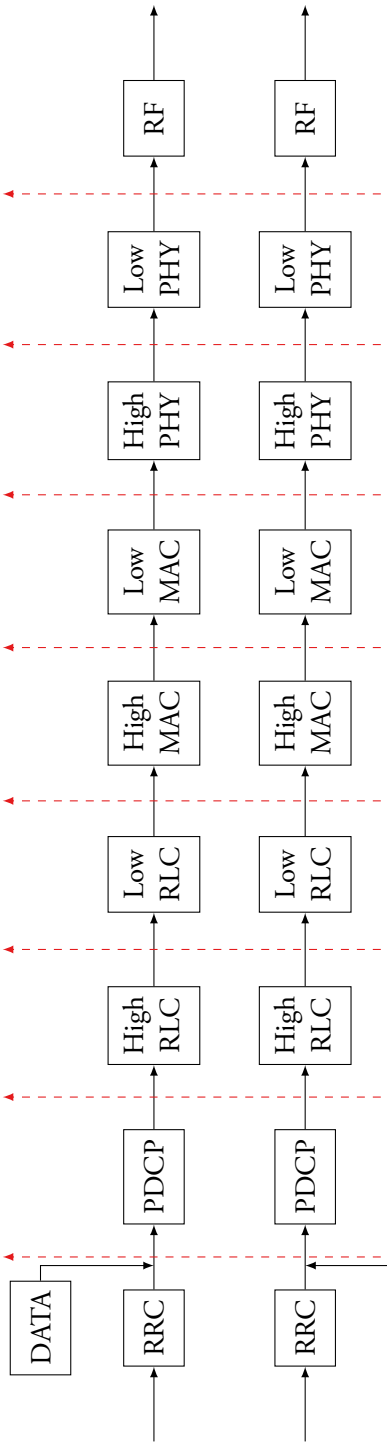


Figure 1.2: 3rd Generation Partnership Project (3GPP) function split options between central and distributed units.

sors (DSPs), and inching towards real-time baseband processing in a cloud-centric model.

While Cloud RAN augments the telecom domain with enhanced efficiency and economic gains, it simultaneously introduces challenges inherent to the integration of the split architecture and cloud technologies. Cloud RAN adopts the 3rd Generation Partnership Project (3GPP) function split options between centralized and distributed radio access units, as outlined in [1417] and illustrated in Figure 1.2. Within this architecture, the split options determine which radio access functions are allocated to the BBU pool and which remain with the RRH, but imposes strict bandwidth and latency constraints on its fronthaul links, and achieving these becomes a formidable task.

For instance, considering split option 8 from Figure 1.2, when all functions above and including Physical Layer (PHY) are handled by the BBU pool, the fronthaul link demands a bandwidth of 157.3Gb/s and permits a maximum one-way latency of 250μs [1417]. Meeting such a requirement is not merely challenging because of the intermediate network and GPPs infrastructure but also due to the cloud computing methodologies applied for function deployment and access management. The intricacies of these challenges will be further discussed in Chapter 3.

In the context of this thesis, we consider a Cloud RAN system featuring function split option 6, which adopts a split between the MAC and PHY layer functions. Generally, a function split between the MAC and PHY layers demands less network bandwidth than alternative configurations. The split option 6 demands a fronthaul link bandwidth of 5.6Gb/s and maximum one way latency of 250μs, which could be potentially achieved by an Ethernet link. Therefore, our setup incorporates an Ethernet fronthaul link that connects a Massive multiple-input multiple-output (MIMO) RRH to a BBU pool hosted within a GPP cloud infrastructure.

1.2 Cloud Control Systems

Along with the emerging of Industry 4.0, time-sensitive and mission-critical industrial applications such as feedback controllers in cyber-physical systems, are transitioning from their conventional platforms to the cloud. This migration seeks the cloud's economies of scale, as well as benefits from offloading, collaboration, and streamlined software development.

Amidst the wave of cloud migration, Cloud Control Systems (CCSs) have garnered immense interest in the emerging industrial age. A CCS is characterized by a physical plant—like robotic arms, autonomous vehicles, and production lines—whereas

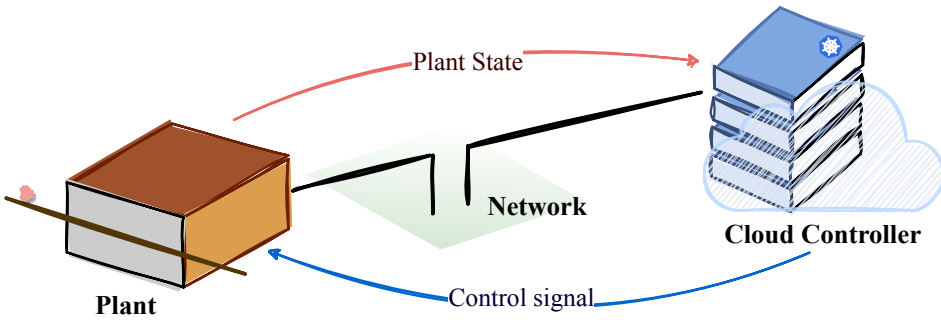


Figure 1.3: An example of cloud control system.

its control mechanism is anchored in the cloud. The control loop of such a system is network-mediated. Complemented by advancements in mobile communication, particularly 5G, these systems are evolving rapidly in the Industry 4.0 landscape, fostering automation across sectors such as manufacturing and mining [Nok23; BSK18].

Figure 1.3 delineates a typical CCS. The control procedure operates periodically, matching the specific plant's demands. In each cycle, the plant communicates its current state to the cloud controller. The latter then processes this data and responds with a pertinent control signal. This signal is then relayed to the plant for the requisite action. This cycle's duration, known as the system's **sampling rate**, determines the intervals between successive plant requests.

A key differentiation between a CCS and a conventional local control system lies in the influence of network latency. This latency can lead to the application of outdated control signals, potentially compromising the process's safety and efficiency. The sampling rate, contingent on the plant model, plays a pivotal role in controller design. A mission-critical control process requires a higher frequency to update its plant states and to get control signals to actuate. Thus, these critical processes require lower network latency as well as efficient computation in the cloud to meet the process dynamics.

Recent literature, such as [Ma+20], has spotlighted dynamic switch solutions between local computing and edge nodes to meet the latency stipulations of cyber-physical systems. Meanwhile, [ODo+18] underscored an industrial cyber-physical construct designed for machine learning tasks, executed via a fog architecture. [Liu22] developed a distributed cloud-based predictive control scheme tailored for networked multiagent systems. This approach introduced a multistep state predictor to counteract the effects of delays inherent in cloud control systems. On the other hand, [Dai+23] transformed a nonlinear Model Predictive Control (MPC) problem into an array of subproblems suitable for parallel computation. To both reduce local computational

burden and enhance efficiency, they crafted a cloud-centric architecture, employing multiple Docker[1] containers to facilitate this parallelization.

Research into harnessing the cloud for real-time systems has surged in recent years. Critical design aspects for cloud-powered real-time services were outlined in [Tsa+10]. [Ska+20] undertook an analytical journey across multiple cloud platforms, proffering strategies to streamline cloud computing in vital control systems. The innovative notion of "control as a service" was discussed in [Ese+15] in the context of autonomous vehicles, integrating a controller tailored to address network imperfections, supported by simulation studies. [Zha+22] devised a two-level hierarchical parallel genetic algorithm, hosted on Spark cloud, for real-time prediction of road traffic flows and proactive setting of traffic signals.

Predominantly, these systems employ a request-response model for message relay between the controller and the plant. A RESTful model, such as HTTP, is frequently used to anchor the controller within a cloud application. The response latency for each request hinges on both the network connectivity and the chosen protocols. Network selection between the plant and cloud controller relies on their spatial relationship, encompassing options like Wi-Fi, mobile networks, Ethernet, or even dedicated optical networks. When controllers are housed in a centralized cloud, application workloads need to traverse the Wide Area Network (WAN). This introduces a layer of unpredictability and intricacies, further complicating the stochastic properties inherent to the cloud controller's response delays.

[1] `https://www.docker.com`

Chapter 2

The Wild – Cloud Computing and Networking

2.1 Cloud Computing

Introduced by John McCarthy in 1961 with the idea of "using computing as public utility, just as the telephone system" [Gar99], cloud computing has emerged as a cornerstone of the modern digital landscape. It offers a transformative approach to accessing software and applications, shifting away from traditional paradigms. By consolidating vast computing and storage capabilities within expansive data centres, cloud computing absolves end-users from the complexities of resource management. Through advanced virtualization technologies, users are offered a shared-resource environment. The pay-as-you-go pricing model further provides flexibility and economic efficiency, sidestepping the need for large initial investments.

The US National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources... rapidly provisioned and released with minimal management effort or service provider interaction" [MG11] and outlines five essential characteristics:

- **On-Demand Self-Service**: Users can independently provision resources.
- **Broad Network Access**: Services are network-accessible from various devices.
- **Resource Pooling**: Users share dynamic resources based on demand.
- **Rapid Elasticity**: Resources adjust automatically based on demand.

- **Measured Service**: Usage is transparently metered for both provider and consumer.

Expanding on the cloud framework, NIST classifies cloud services into three primary models:

- Software-as-a-Service (SaaS): This model provides users access to the provider's applications running on a cloud infrastructure. Here, users interact with the application via client interfaces such as web browsers, but don't manage or control the underlying cloud infrastructure.
- Platform-as-a-Service (PaaS): Tailored for developers, this model offers a platform allowing consumers to develop, run, and manage applications without delving into the complexities of building and maintaining the infrastructure. While the cloud provider manages the network, servers, and storage, consumers have the flexibility to modify their application environment.
- Infrastructure-as-a-Service (IaaS): The most flexible of the three, IaaS offers consumers the tools to provision processing, storage, and other fundamental computational resources. Here, consumers can deploy and run arbitrary software, including operating systems and applications, having control over aspects like storage, deployed applications, and certain networking components, but not the underlying infrastructure.

Building upon the foundations of cloud computing, newer paradigms like edge computing and fog computing have also emerged. These paradigms prioritize speed, efficiency, and privacy, distinguished by their proximity to consumers and inherent distributed nature.

2.1.1 Public Cloud

Public cloud is one of the cloud deployment models defined by NIST[MG11]. It is also commonly referred to centralized cloud, standing in contrast to the distributed nature of edge and fog computing. This cloud infrastructure is available for widespread use by the public and can be owned, operated, and managed by various entities, whether they're businesses, academic institutions, government bodies, or a mix of these. Currently, the public cloud landscape is dominated by giants such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Collectively, they hold approximately 75% of the global market share [Ric]. These titans offer an extensive array of services tailored to multiple models, supporting a vast range of applications, from Internet of Things (IoT) and gaming to machine learning. In the context

of this thesis, we primarily employed AWS services when there is a need to integrate the public cloud into our test-bed.

2.1.2 Edge, Fog and Mobile Edge Cloud

Emerging from the foundational concepts of cloud computing, both edge and fog computing were conceived to serve time-critical processes. Their primary objective is to enhance the responsiveness of cloud applications. They achieve this by situating services within smaller, decentralized cloud infrastructures located closer to users. While this facilitates speedier interactions, it comes with the trade-off of having comparatively constrained computing capabilities in contrast to centralized clouds.

The edge computing paradigm positions shared computational resources at the network's edge, closer to end-users. This setup ensures resources are coordinated in a cloud-like fashion. By positioning controllers at the edge, they can leverage cloud benefits without enduring the latency and unpredictabilities associated with traversing vast Internet expanses.

Edge and fog computing are closely related, both originating from cloud computing principles and featuring distributed processing capabilities. In numerous systems, edge and fog nodes act synergistically with centralized clouds, amplifying the advantages each brings to the table [ODo+18; MRB18]. However, there's an evident drive to exclusively utilize either the edge or fog layers in specific systems, aiming to fully harness the boon of reduced latency [Pan+17; Ma+20; Hao+18].

Our understanding of edge and fog is anchored in the definitions postulated by NIST and European Telecommunications Standards Institute (ETSI). The NIST characterizes the fog paradigm as a layered and distributed model. This model comprises both physical and virtual fog nodes, strategically positioned along the data trajectory from end-user devices to centralized cloud services [Ior+18]. Thus, fog computing emerges as a cooperative computational paradigm, seamlessly integrating centralized clouds with other essential computing elements strewn across the user-cloud data pathway.

Conversely, edge computing is rooted in the principles of Content Delivery Network (CDN), which prioritizes delivering video content via edge servers located near users, typically within Internet Service Providers (ISPs) or network operator domains. Expanding upon this, edge nodes provide computing capabilities for application deployment and are inclined to incorporate virtualization technologies, a hallmark inherited from cloud computing [MMB18].

Both NIST [MMB18] and ETSI [Hu+15] have also placed considerable emphasis on the concept of Mobile Edge Computing (MEC). This approach promotes an IT ser-

vice environment at the edge of mobile networks within the RAN, ensuring closeness to mobile users. MEC fuses the cloud computing benefits with distinctive features like low latency, cost efficiency, and real-time processing. This unique blend offers Mobile Network Operators (MNOs) a golden opportunity to deliver mission-critical services to their subscribers.

For the research detailed in this thesis, our focus has been predominantly on employing edge computing and mobile edge computing for system deployment, eschewing the need for extended data pathways of an application to a centralized cloud.

2.1.3 Industry 4.0 and Cloud Computing

The rapid digital transformation powered by Industry 4.0 is steering the integration of cloud computing into manufacturing systems, unlocking the potential for "infinite" computing resources and storage capacity. This integration is reshaping industrial real-time applications like controllers and schedulers for cyber-physical systems, enabling them to offload computing tasks to clouds. The manufacturing industry stands to gain substantially from this new paradigm, as the economic model and flexibility of cloud computing translate into efficiency and cost-effectiveness [Xu12].

In addition, the emergence of the Industrial Internet of Things (IIoT) as a vital part of the industrial revolution has reinforced the importance of cloud computing in modern industrial systems. As on-board processing on IIoT devices is often limited, offloading data analysis and computing from a large-scale deployment of IIoT applications to the cloud is recognized as an optimal solution for the manufacturing sectors [Buy+16; Geo+16; PL16]. This approach, integrating IoT applications with cloud services, has been reported to lead to significant productivity improvements in manufacturing plants [Cis19]. The conjunction of IIoT and cloud computing paints a compelling picture of the future, where interconnected devices and data-driven insights drive the next wave of industrial innovation.

2.1.4 EIT Kubernetes Cluster

As highlighted in Section 2.1.2, the research contained within this thesis predominantly leverages edge and mobile edge computing. Primarily, the applications are deployed on the EIT edge cluster—a seven-node bare-metal Kubernetes[1] cluster located within the EIT wireless systems and application lab. Significantly, this cluster shares the same Local Area Network (LAN) as the end-users of the deployed applica-

[1] `https://kubernetes.io`

tions. At the time of this writing, each node operates on the Ubuntu-jammy operating system, featuring the Linux kernel version 5.15.0-69-generic.

This cluster, being a bare-metal one, uses distinct physical machines as worker **Nodes**. Consequently, we refrain from deploying virtual machines or networks for the cluster. Nevertheless, every application on this cluster is containerized using Docker and is orchestrated and maintained by Kubernetes.

Kubernetes stands as one of the leading open-source orchestration platforms designed for deploying, orchestrating, and scaling containerized applications. Within this system, each Node can accommodate multiple **Pods**, comprising one or multiple containers with shared storage and network resources. A Pod represents the most elemental deployable computing units in Kubernetes. While an application in Kubernetes is manifested as one or multiple pods, and it is exposed to the network as a **Service**. This ensures the application remains accessible for interactions with the end-users.

The EIT Kubernetes cluster serves as the primary infrastructure for deploying cloud/edge applications discussed in this thesis and other related works. Given that all prominent cloud providers now offer Kubernetes services (e.g., AWS EKS, Google Kubernetes Engine, Azure Kubernetes Service), it becomes seamless to deploy our edge applications onto centralized or public clouds, enabling evaluations and validations in diverse environments.

2.2 Wild Networks

By "wild networks", we are referring to the actual networks employed in a system or a testbed, not the idealized models found in simulations. While network simulators are indispensable tools for researchers, they often fall short of accurately representing the real-world networks that operate under a variety of daily scenarios. After all, no model can perfectly capture every nuance and variable.

The networks between a local system and the cloud are often not consistently reliable. A myriad of factors contribute to this unreliability, including the nature of the transmission media, the infrastructure's robustness, the communication protocols used across all layers, and the interference from other user traffic.

In this chapter, we focus primarily on those factors that have a direct impact on the research problems tackled in this thesis. These considerations were at the forefront of our investigation into cloud integrated systems. They shaped our understanding of how these systems operate within the confines of real-world networks and guided our efforts to devise solutions that take into account these complex, often unpredictable

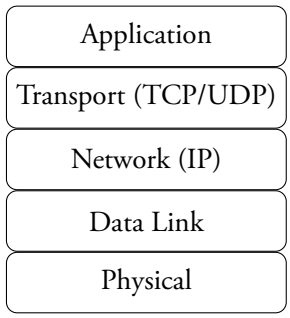


Figure 2.1: Network protocol stack in TCP/IP model.

environments.

2.2.1 Communication Protocols of Cloud applications

In the scenario of a cloud integrated system, remote computational tasks often necessitate significant alterations in deployment methodologies to adhere to the cloud paradigm as cloud applications or services. This transition compels the implementation of a full computer networking stack for the cloud application and demands that the system's data path traverse the IP network.

Such a configuration necessitates the application of the full network stack implementation derived from the TCP/IP model, as illustrated in Figure 2.1. The selection of transport and application protocols in the model is largely contingent upon the Quality of Service (QoS) requisites of the system. While the Internet protocol suite, is commonly recognized as TCP/IP, which has dominated the Internet landscape for decades, other transport protocols like UDP and QUIC are also vital components within the realm of contemporary web services.

However, it's crucial to note that higher layer protocols can potentially introduce disruptions to the QoS of the transformed cloud applications within our systems. These protocols typically favour intricate transmission control mechanisms aimed at ensuring the reliability of data transmission. Therefore, a comprehensive understanding of the pros and cons of these protocols is vital when selecting among them.

In the following section, we offer an overview of the transport layer and application layer protocols that we have evaluated and employed in our cloud applications during the deployment of our cloud integrated systems. This discussion aims to illuminate the considerations and challenges inherent in this critical aspect of system deployment.

TCP, UDP and QUIC

As mentioned earlier, TCP is the most commonly used transport layer protocols for today's web services, as well as cloud applications and services. TCP offers several advantages that make it an ideal choice for many network communication applications.

TCP's strength lies in its reliability through its use of **acknowledgments** and **automatic retransmission** of lost or corrupted data. TCP also employs **flow control** and **congestion control** algorithms, which adjust the data transmission rate to the recipient's processing capacity and network traffic conditions, thereby maintaining stability and predictability. TCP's built-in **handshake** procedure further ensures readiness for communication and allows for graceful connection termination, thus preventing abrupt disconnections and potential data loss. Consequently, TCP's reliability, efficiency, and robust connection management make it a highly preferred protocol in network communications.

Contrary to TCP, UDP operates on a simpler mechanism and does not establish a connection prior to data transmission, thereby offering faster speeds. It does not provide reliability features such as acknowledgment, retransmission or flow control. Those features make UDP suitable for application where speed and low latency are more important than absolute reliability as their QoS.

QUIC, a transport layer protocol leveraging UDP, was first proposed by Google in 2012 as an innovative substitute for TCP in deploying modern web applications. Despite operating over UDP, QUIC manages to reproduce TCP's functionalities, employing similar but uniquely differentiated mechanisms. The major functions offered by QUIC, contrasted with those of TCP, are succinctly encapsulated in the subsequent Table 2.1.

The following provides detailed insights and clarifications for each of these functionalities in Table 2.1.

Handshakes Establishing a TCP connection necessitates the TCP client to await an acknowledgment from the server to its SYN packet before data transmission can commence, thus requiring a minimum of one Round Trip Time (RTT) to establish the connection. When secure transmission is called for, a cryptographic handshake is also required, contributing an additional two RTTs to the connection establishment preceding data transmission. Conversely, QUIC employs an integrated cryptographic and transport handshake, which operates on a 1-RTT or 0-RTT basis, contingent on whether a QUIC connection is being initiated or resumed. A more detailed depiction of the cryptographic and transport handshakes integral to both protocols is provided

Table 2.1: Comparison of major functionalities in TCP and QUIC, summarize from [IT21; IS21; For07]

Functionality	TCP	QUIC
Handshakes	Three-way handshake	1-RTT or 0-RTT handshake
Multiplexing	Depending on different application protocols	Allows arbitrary number of concurrent streams.
Congesting control	Slow start and congestion avoidance policies	Similar to TCP congestion control mechanism with larger minimum congestion window
Flow control	Window-based flow control	Offset-based flow control for both streams and connections
Loss detection	Retransmission Timeout (RTO)	Probe Timeout (PTO)
Retransmission	The lost packet is retransmitted with the same sequence number	The frames in a lost packet will be put in a new packet for retransmission with new packet number, the lost packet is not retransmitted.
Crypto. Implementation	No cryptographic implementation, need an extra TLS layer	Embedded TLS implementation on QUIC layer

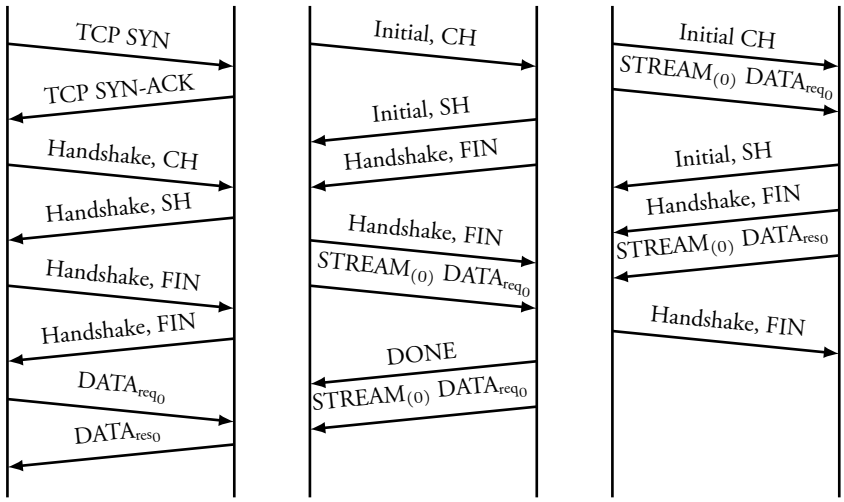
in Figure 2.2.

Multiplexing The multiplexing mechanism within TCP relies on the application protocols built atop it, as elucidated in Section 2.2.1. In contrast, QUIC inherently supports multiplexing by allowing a single connection to accommodate multiple concurrent streams.

Congestion control The congestion control mechanism in QUIC mirrors that of TCP, but with a notable variation. While TCP sets the minimum congestion window to one packet, QUIC recommends the use of two packets.

Flow control TCP employs window-based flow control where both sender and receiver adjust their respective send and receive window sizes to manage data flow. On the other hand, QUIC introduces both connection-level and stream-level flow control. In this setup, the receiver announces the maximum number of bytes it is ready to receive either for a specific stream or for the entire connection, providing a more fine-grained control compared to TCP's approach.

Loss detection TCP employs a single Retransmission Timeout (RTO) for each connection, considering a packet lost if its acknowledgment isn't received before the timeout expires. QUIC, in contrast, calculates a Probe Timeout (PTO) for each packet



(a) HTTP1 (TCP) (b) HTTP3 1-RTT (QUIC) (c) HTTP3 0-RTT (QUIC)

Figure 2.2: TCP and QUIC transport and cryptographic handshakes for connection initiation.

space. An interesting nuance here is that timer expiration in QUIC doesn't necessarily signal packet loss. Rather, it initiates the sending of one or two probe datagrams, facilitating a recovery mechanism from potential loss.

Retransmission Unlike TCP, where packet numbers are assigned based on sequence and retransmissions carry the same sequence number, QUIC takes a different approach. QUIC employs strictly monotonically increasing packet numbers, so a packet with a higher number is sent later. Consequently, if a packet gets lost in QUIC, the protocol places all frames from the lost packet into a new packet for retransmission, instead of retransmitting the original packet itself.

Cryptographic implementation QUIC enhances secure transmission by incorporating a TLS implementation within its own layer, thus eliminating the need for an additional TLS layer, which is a requirement in TCP due to its lack of embedded cryptographic implementation.

HTTP

The foundation of data communication for the World Wide Web is the application protocol HTTP, which first made its appearance in the early 1990s. Operating in a

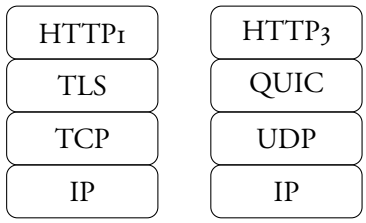


Figure 2.3: Application, transport and network layers of HTTP1 and HTTP3.

RESTful style, it adopts a client-server model in its request-response mechanism. The protocol's secure extension, HTTPS, is in use by approximately 83.6% of all websites as of August 2023 [Arc], with HTTPS requests reaching a staggering 96.7% by July 2023 [Web].

Over the past decades, HTTP has undergone multiple iterations and improvements, resulting in three major standardized versions being deployed on the web - HTTP1.1, HTTP2, and HTTP3.

HTTP1.1 Standardization completed in 1999, HTTP1.1 maintained its reign over the World Wide Web for several years [Nie+99]. To enable multiplexing, an HTTP1.1 client was required to establish multiple TCP connections. However, with the surge in traffic load, HTTP1.1 began to falter in delivering a satisfactory web browsing experience. Issues such as Head-Of-Line (HOL) blocking, expensive connection establishment for multiplexing, and slow web page loading speeds began to surface.

HTTP2 To counter the challenges of HTTP1.1, HTTP2 was announced and standardized in 2015 [BPT15]. With its introduction of efficient multiplexing that facilitated concurrent streams over a single TCP connection, HTTP2 brought about significant improvements. Nonetheless, it inherited the HOL blocking problem from its predecessor due to its reliance on TCP, which ensures packet receipt in the exact sequence of their transmission. This meant that when a packet was lost, the subsequent packets were forced to wait until the lost packet was retransmitted.

HTTP3 Fast forward to 2022, and we have the standard release of HTTP3 [Bis22], also known as HTTP over QUIC. Unlike the previous versions of HTTP that were dependent on a separate TLS layer for secured data transmission, as illustrated in Figure 2.3, HTTP3 entrusts all functionalities to QUIC, including cryptographic and transport handshakes, as well as multiplexing. Similar to HTTP2, this protocol also supports concurrent data streams over a single QUIC/UDP connection. However, it

eliminates the HOL blocking problem owing to its UDP-based nature and distinct retransmission mechanism. This, in turn, mitigates the connection establishment issues seen in previous versions, reduces network latency, and improves overall performance.

Throughout this thesis, HTTP1.1 is predominantly used in our cloud applications, considering our applications do not involve large-sized page loading. Additionally, we have also implemented some systems in HTTP3 to evaluate its performance against HTTP1.1.

MQTT

The HTTP family, although prevalent, is not the sole suite of application protocols employed within the realm of cloud computing. The Message Queue Telemetry Transport (MQTT) protocol, an OASIS standard, is extensively adopted in cloud applications. This protocol typically operates over TCP, but there are variants that utilize QUIC [SM23] or even UDP [Zav22] for transmission. MQTT, being a lightweight protocol designed for message queueing services, leverages a publish-subscribe model, making it particularly suited for IoT communications. Some implementations of our cloud integrated systems also incorporate MQTT due to its advantageous properties.

2.2.2 5G and Network Systems

The 5G offers substantial enhancements over its predecessors. Deployed globally since 2019, it shifts the focus from enhancing everyday user experiences like voice calls, messaging, web browsing, and video streaming, to providing benefits for both consumers and a vast array of industries. 5G delivers lower latency, greater capacity, and higher reliability, laying the foundation for three main categories of use cases: Ultra-Reliable and Low-Latency Communication (URLLC), Massive Machine Type Communication (mMTC), and Enhanced Mobile Broadband (eMBB), all of which are delineated in Figure 2.4.

The basic architecture of mobile networks has remained fundamentally consistent across generations, despite significant advancements in technologies. This architecture primarily comprises the RAN and the Core Network (CN). The RAN, facilitated by RBSs, directly communicates with User Equipment (UE), establishing and managing radio connections. Conversely, the CN functions as the mobile network's backbone, handling data routing, user mobility, session management, and connectivity to external networks, such as the Internet. In the subsequent sections, we delve into some pivotal elements and technologies pertinent to this thesis.

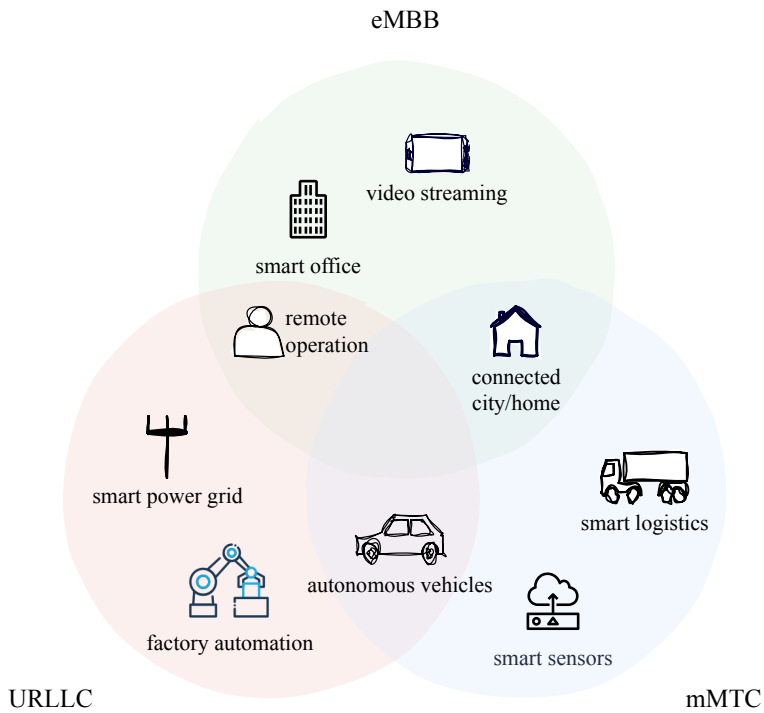


Figure 2.4: Three main categories of 5G use cases and example applications [Tey+17].

Radio Access Network

The RAN serves as the gateway for mobile users to connect to mobile networks. Radio Base Stations (RBSs), fundamental components of RAN, are typically composed of two main parts: the radio heads and BBUs, linked through fibre optic networks. The radio access technology in 5G is designated as New Radio (NR), which covers an expansive range of frequency bands, from the traditional GSM (2G) bands to the advanced millimetre wave. Massive MIMO is a standout technology in 5G NR, delivering significant improvements in spectral efficiency. At the same time, Cloud RAN exemplifies the integration of cloud networking in the 5G landscape. Both of these transformative technologies are discussed in depth in Part II when a radio resource problem under industrial scenario is discussed.

Massive MIMO Massive MIMO builds upon the foundational idea of Multiuser MIMO, which centres on serving multiple terminals using the same time-frequency resources concurrently. As a pivotal component of the 5G RAN, Massive MIMO is geared towards handling the rapid growth of traffic in mobile networks by dra-

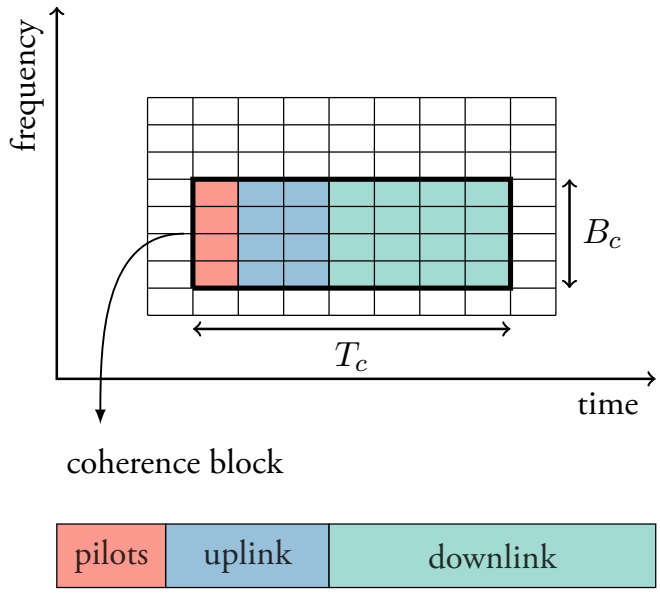


Figure 2.5: An example of a coherence block in Massive MIMO time-frequency spectrum.

matically enhancing system capacity. By deploying a considerably larger number of antenna arrays, Massive MIMO can achieve elevated spectral efficiencies, ensuring seamless service to a vast number of mobile devices simultaneously.

The Massive MIMO system and its parameters discussed in this thesis is based on the LuMaMi testbed, which is a 100-antenna system developed in the department of EIT, LTH [Vie+14]. The system functions using Time Division Duplex (TDD) and Orthogonal Frequency-Division Multiplexing (OFDM) modulation. For the communication of UEs, a RBS needs to understand the channel characteristics by estimating the Channel State Information (CSI) from pilots. In a TDD operation, the RBS acquires knowledge of the uplink channel through uplink pilots, which subsequently provides a valid estimation for the downlink due to channel reciprocity [Mar+16].

In a single Massive MIMO system, the time-frequency space is segmented into coherence blocks. Each coherence block represents the maximum time duration in which the channel appears time-invariant and maintains a near-constant frequency response. This duration, termed the coherence time, depends on the carrier frequency of the channel as well as the terminal's movement speed. A faster moving UE results in a reduced coherence time, necessitating more frequent channel estimations for the same UE. Coherence blocks accommodate uplink data, downlink data, and uplink pilot transmissions. A detailed representation of a coherence block can be found in Figure 2.5 and is further elaborated upon in [Mar+16].

As highlighted earlier, uplink pilots allow the RBS to ascertain the CSI for each UE. This CSI then aids the RBS in the encoding processes essential for handling incoming and outgoing data. Therefore, an uplink pilot is imperative for a UE to successfully transfer data. A challenge arises because the total number of orthogonal pilots is restricted by the channel coherence interval. Meanwhile, the number of UEs seeking data transmission might significantly outnumber the available pilots within a coherence interval. This discrepancy is even more pronounced in Massive MIMO systems, which aim to serve numerous UEs simultaneously. This leads to the pivotal challenge of selecting optimal pilot access techniques that factor in the specific traffic conditions of various scenarios to prevent pilot conflicts or contamination. The dilemma of pilot accessing is essentially a resource allocation challenge at the MAC layer and is explored in Part II of this thesis.

Core Network

The CN serves as the backbone of the mobile network, be responsible for user connectivity, mobility management, data routing, and facilitates access to external networks like the Internet. In the realm of 5G, the CN has transitioned to a software-driven, service-based architecture. This shift allows the 5G core to be seamlessly deployed on cloud-native, container-based platforms where all core functionalities manifest as microservices. This software-centric approach not only enhances the scalability of the 5G core but also bolsters its resilience against faults [Hat].

Presently, 5G mobile networks employ two distinct types of CNs: Non Stand-Alone (NSA) and Stand-Alone (SA). NSA is prevalent in current commercial 5G deployments. It features a 5G core that coexists with the Long Term Evolution (LTE) core, enabling user access through both LTE and 5G RANs. Such an arrangement expedites the rollout of new 5G services while optimizing the utility of pre-existing LTE infrastructures. Conversely, the SA core exclusively interfaces with 5G RANs and represents the ideal 5G mobile network setup. However, its adoption necessitates considerable investments in new network infrastructures, rendering it less attractive for immediate commercial deployment.

2.2.3 LTH 5G RBS

This section provides an overview of the 5G RBS situated in the Wireless System and Application Lab within the Department of EIT, LTH. The RBS discussed herein facilitates the 5G network essential for all test-beds and experiments associated with this thesis involving 5G.

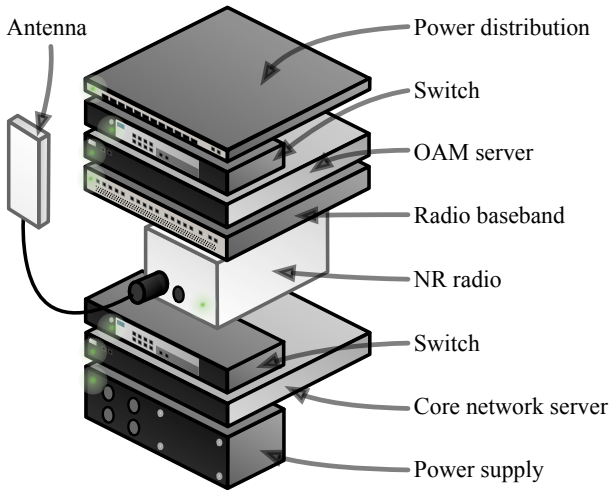


Figure 2.6: The anatomy of the LTH 5G deployment in stage 1.

Established as a research initiative in collaboration with Ericsson in 2021, the LTH 5G network featured in this thesis is a mid-band 5G NR SA RBS. This collaboration has granted us the flexibility to reconfigure radio parameters, and we've elected to integrate an open-source core network.

Over the past two years, the RBS underwent two significant evolutionary stages:

1. **Initial Phase:** Initially, the system was calibrated to function within the NR band n3 (1800 MHz). This configuration facilitated the 5G network as elaborated in Chapter 7. The component arrangement of this deployment can be referenced in Figure 2.6. Key components include a power distributor, a switch for unit interconnection, the Operations, Administration and Management (OAM) server for unit management, the BBU (Ericsson Baseband 6630), and the NR Frequency-Division Duplexing (FDD) radio head (Ericsson Radio 2219) connected to a single passive indoor antenna. The system also incorporated a CN server running on the Ubuntu Operating System, facilitating 5G UEs access to mobile applications and services. This server functioned on Open5GS[2], an open-source C-Lang implementation of 5G core and Evolved Packet Core (EPC), specifically utilizing the SA 5G core version.

2. **Advanced Phase:** Detailed in Figure 2.7, the subsequent phase witnessed the system's recalibration to the NR band n78 (3500 MHz). This iteration is utilized in the evaluation in Chapter 8, featured two Ericsson TDD Radio Dots (Ericsson Dot 4479) connected to the indoor radio unit (Ericsson IRU 1648).

[2] `https://open5gs.org/`

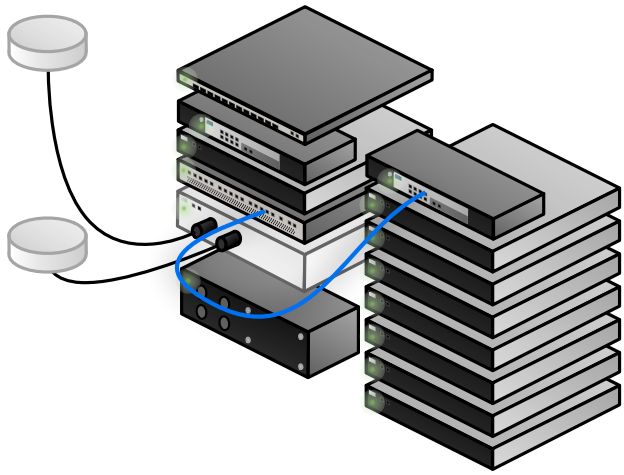


Figure 2.7: The anatomy of the LTH 5G deployment in stage 2.

Together, these components constituted a distinct 5G cell that could extend its coverage both within and outside the lab with more radio dots installed in the future. The BBU and associated components persisted from the initial phase. However, the CN underwent a transformation. While Open5GS continued to form its backbone, all network functions were now manifested as microservices within the edge EIT Kubernetes cluster (detailed in Section 2.1.4), as portrayed on the right side of Figure 2.7. This restructuring resulted in a core that was intricately integrated with mobile edge applications housed within the same Kubernetes cluster.

To round off this chapter, it's worth noting that while our narrative heavily leans on real-world networks and practical network and cloud infrastructures, the value of simulations in our research remains undiminished. They provide a focused environment, allowing us to address specific research issues without being mired in the intricacies of a multi-faceted network system. For aspects of this thesis that touch upon infrastructures beyond our access, such as massive MIMO, we still turn to simulations as a dependable methodological approach.

Chapter 3

Challenges

This thesis primarily focuses on two forms of cloud integrated systems: Cloud RAN and Cloud Control System. While the prior chapter provided an overview and key features of the cloud and the expansive networks pertinent to these systems, this chapter delves into the specific challenges associated with these two types of systems, which this thesis aims to address.

It's vital to underscore that the backdrop of this thesis is framed by the industrial digital transformation. The evaluation of Cloud RAN and CCS is intimately tied to an industrial context. Assessing the performance of these systems in a vacuum, disconnected from the practical environment in which they operate, would be inadequate.

The introduction of the Industry 4.0 concept at the 2011 Hannover Trade Fair [AG] marked a significant shift. Often referred to as the Fourth Industrial Revolution, it signified Germany's transition towards a modernized manufacturing paradigm, where traditional methodologies harmoniously coexist with emerging technologies like IoT, 5G, Artificial Intelligence (AI), and cloud computing. Drawing from this transformative context, this thesis delves into the Cloud RAN and CCSs. These frameworks aim to fuse the strengths of mobile networks and cloud computing with the rapidly evolving industrial landscape. In the ensuing sections, we delve into the challenges and requirements of cloud integrated industrial systems, emphasizing the interplay of wild networks (typically mobile networks) with the cloud.

3.1 Mobile Networks and Cloud RAN for Industrial Communication

In the framework of Industry 4.0, wireless communication for factory automation is gaining momentum. Such automation demands networks that not only offer high reliability, capacity, and throughput but also ensure low latency, coupled with adherence to security and safety standards, to cater to diverse application needs [ZSG17].

Traditionally, factory automation has leaned on wired solutions like Controller Area Network (CAN), Modbus, and industrial Ethernet. While these offer reliability, they restrict device mobility and pose significant costs associated with cable installation and upkeep. Though wireless networks address these limitations, many, especially those in unlicensed spectrums, grapple to match the performance standards of their wired counterparts. This disparity has spurred active research into optimizing industrial wireless standards.

With the advent of 5G, a transformative alternative emerges, addressing many limitations inherent to conventional wireless networks [GLA17]. 5G satisfies numerous industrial automation requisites, from ultra-reliability to low latency [Mat+16], while simultaneously enhancing mobility and adaptability in production systems.

Nonetheless, integrating wireless solutions in an industrial context presents challenges. A predominant concern is the sharing of radio spectrum resources. Despite manufacturers' inclinations to utilize licensed spectrums, the myriad industrial devices vying for ultra-reliable, low-latency data communication can strain these resources. Absent effective RAN management, this can precipitate resource shortages in industrial communications, potentially leading to operational failures.

As a leading contender for 5G RAN architectures and future advancements, the integration of Cloud RAN into industrial settings amplifies the challenges tied to radio resource sharing. While it promises notable economic advantages—such as reduced deployment and maintenance expenses—it's confronted by inherent limitations due to its unique architecture and infrastructure setup.

Central to the radio resource management is its operation at the MAC layer, situated within the BBU pool. The prompt reaction of the BBU pool to the resource signalling initiated by UE becomes crucial. However, the inherent design of Cloud RAN poses hurdles, often rendering it less effective in this domain compared to conventional RBSs.

The academia landscape showcases numerous studies addressing resource allocation, especially for low-latency communication services within Cloud RAN across varying

contexts. For instance, [Zha+20] directs attention to minimizing energy consumption for computational tasks within a mobile edge cloud-empowered Cloud RAN system. Meanwhile, [WZM16] introduces an energy-conserving joint resource scheduling strategy specifically for Cloud RAN systems. Research endeavours like those in [FDA17] and [MWY17] leverage distributed allocation algorithms to pare down response or computational latency within Cloud RAN frameworks. [Moo+21] explores a user-centric resource allocation in OFDMA-based Cloud RAN, enabling users to select RRHs groups based on their delay specifications. [Sha+22] develops a Mixed Integer Linear Programming solution that jointly allocates radio resource blocks and computing assets like CPU processing time to minimize a Cloud RAN system's power usage. Employing a different approach, [ITC22] introduces a deep reinforcement learning model aiming to derive an optimal control strategy for turning the RRHs on or off. This optimizes the trade-off between spectral and energy efficiency in a Cloud RAN system. Lastly, [Oca+23] integrates the BBU pool of Cloud RAN with MEC servers within a GPP framework, introducing a CPU sharing mechanism to diminish the performance effects of resource sharing on BBU.

Yet, with the commercial deployment of Cloud RAN still in its nascent stages, and its application in industrial communication even more limited, there remains an imperative to further delve into its potential and suitability for industrial communication.

3.2 Cloud Integrated Industrial Systems

Beyond network reliability, integrating cloud computing into industrial systems presents challenges. These arise primarily from the cloud's unpredictable execution environment, which contrasts with the low-latency and ultra-reliability demanded by industrial applications.

3.2.1 Virtualization and Containerization

Virtualization is the key technology leveraged by cloud computing that allows the physical resources such as CPU, memory, storage and network being pooled and shared by multiple tenants in the cloud. Virtualization brings the benefits of higher flexibility, faster resource provision, cost reduction and higher resource utilization, however, at the cost of performance suffering. As it adds abstraction layers on top of physical machine, yielding a longer path of the workloads than that in a bare-metal server. [Ha+16] has shown that the size of Virtual Machines (VMs) can have big impact on I/O performance, which makes it crucial to choose the appropriate favours of VMs for cloud applications, especially Big Data application. [Dre08] details the cost

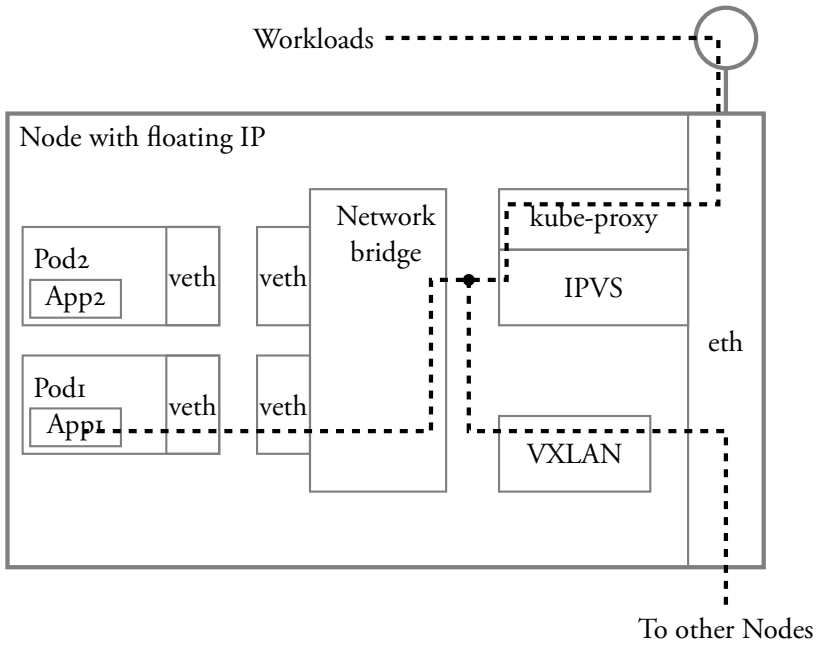


Figure 3.1: An example of workload path to reach a service in a typical cloud environment through Kubernetes NodePort deployment. Figure from [Lar+20].

of virtualization and highlighted the changes a program experiences when been deployed on a VM. [WN10] also shows large delay variations and unstable TCP/UDP throughput caused by virtualization on AWS EC2 instances. All of these above will add to large network latency and unreliability to a cloud application, which is the least favourite to an industrial system.

In addition, containerization technologies, like Docker, introduce overheads. Although a container functions as the most basic computational unit in cloud-native deployments and shares the Linux kernel with its host, it perceives itself as an independent machine. The host therefore must encapsulate incoming packets with IP headers to guide them to the intended container.

3.2.2 Cloud-Native Networking

Cloud environments also impact access control and internal networking of cloud applications. Consider an application orchestrated by Kubernetes. As discussed in Section 2.1.4, Kubernetes encapsulates containerized apps within Pods and exposes them as Services across a multi-node cluster. When services are deployed atop Kubernetes, networking functionalities are provided by Container Network Interface (CNI). This

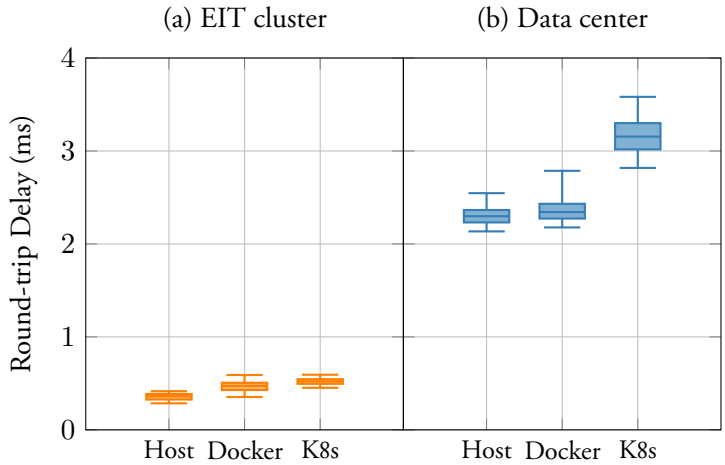


Figure 3.2: The UDP latency measurements on response time to different type of services deployments, 1.5% outliers removed

arrangement introduces additional hops for workloads to access a service endpoint. [Lar+20] illustrates in Figure 3.1 how traffic is routed to a service endpoint within a Kubernetes node, epscifillcay when using a NodePort[1] service type. And when functions are interconnected but located on different nodes, workloads navigate via the Virtual Extensible LAN (VXLAN) to the next endpoint.

These configurations introduce performance overheads by adding extra layers to workload paths, potentially compromising the ultra-low latency needed for industrial applications. This results in unpredictable response delays from cloud applications, which can degrade system performance.

3.2.3 The Delays in Cloud Integrated Systems – A Clarification

As previously emphasized, a key focus of this thesis is on the "delays" encountered in cloud integrated systems and their impact on system performance. Adopting the traditional definition of computer network delays —encompassing transmission, propagation, queueing, and processing delays —for modelling end-to-end delays in such systems is challenging. This is due to the cloud-hosted segment of the system obscuring its infrastructure from the other end. In our research, we concentrated solely on measurable delays. Within the context of a system "controlled over the cloud" as detailed in Chapter 1, our attention was directed to the delays observed by the control object. Specifically, this refers to the duration the control object awaits a response

[1] `https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types`

from the cloud controller after initiating a service request. Using a system with a RESTful communication architecture between the control object and the controller as an example, the delay in focus is the response time of the cloud service.

To provide a clearer perspective on the delay nuances introduced by the mentioned network configurations, containerization techniques, and Kubernetes deployments, we turned to empirical measures. Figure 3.2 presents our findings based on the response times of a rudimentary "UDP PING" application. We have performed the measurement when the application is:

(a) Deployed within the EIT Kubernetes cluster, co-located on the same LAN as the client machine and connected with a direct 1Gps Ethernet link.

(b) Hosted within the Ericsson Research Data Center (ERDC) in Lund, albeit within the same urban locale as the client, where the network in between is a combination of 1Gps Ethernet supplemented with WAN.

Within these locations, the application's performance was gauged under three distinct deployment paradigms:

- Directly on a host machine, bare-metal in case (a) and VM in case (b).
- Within an isolated Docker container.
- Housed within a Kubernetes Pod, showcased via a NodePort Service.

The visual representation in Figure 3.2 tells a compelling story. With the consistency of the application, there is a marked increase in both the mean response time and its variability as we transition from a native host deployment to a Docker container and, subsequently, to a Kubernetes Pod. Notably, the Kubernetes arrangement within the confines of a cloud data centre demonstrates heightened overheads relative to its bare-metal counterpart. This increased latency owes much to the cloud data centre's inherent virtual network structure, an entity that largely remains outside the direct purview and management of application overseers.

Chapter 4

Thesis Outline & Contributions

The primary aim of this thesis is to delve into two specific types of cloud integrated systems: Cloud RAN and CCS. Our objectives are to evaluate the viability of implementing these systems within a realistic setting, assess their operational efficiency, and explore avenues for performance enhancement. Crucially, we accept that the "wild" environment-primarily composed of the network and cloud infrastructure-remains untouched, tailored neither for our specific systems. Instead, our strategy revolves around modifying the system to fit within this environment, while also minimizing performance setbacks attributed to said environment. To summarize, the primary contributions of this thesis are threefold, each corresponding to a distinct topic:

- Radio resource allocation in Cloud RAN for industrial scenarios: Focusing on radio resource allocation, this part investigates the feasibilities of employing a single-cell massive MIMO RBS and Cloud RAN within an industrial setting. The aim is to facilitate mobile communications for diverse categories of industrial units, meanwhile satisfying their QoS requirements.
  - Chapter 5 initiates our exploration by considering a traditional RAN system within the industrial context. Here, we introduce a MAC layer slicing strategy designed to manage the radio allocation challenges associated with two types of industrial UEs.
  - Chapter 6 transitions to the Cloud RAN system within a similar industrial context, focusing on the practicalities and challenges of its deployment, especially in meeting the QoS demands of time-sensitive UEs.
- Implementation of CCSs: This part showcases real-world deployments of CCSs across diverse networks, including Ethernet and 5G, as well as in various cloud

execution environments like edge and public cloud platforms. Additionally, it examines the influence of network configurations and protocols on the performance of CCSs.

  - Chapter 7 demonstrates the deployment of a CCS over a 5G network, with the controller situated on an edge node of this 5G network. It also examines the impact of different radio configurations on the system performances.
  - Chapter 8 investigates the implications of various transport layer protocols on the CCS's performance across diverse network and cloud execution environments.

- A framework for achieving punctuality in time-critical cloud integrated systems: Drawing from our prior studies on cloud integrated systems, this part introduces a novel framework called "Punctual Cloud", designed for implementing cloud integrated systems using a microservice architecture. The framework aims to mitigate the effects of system induced delays by ensuring that each response from the cloud controller is executed by the control object promptly and within a defined time frame.

  - Chapter 9 integrates this framework into a Cloud RAN system within the industrial context, aiming to enhance the outcomes of the radio resource allocation process.
  - Chapter 10 employs the framework within a CCS, striving to bolster the system's resilience against the prolonged delays inherent in both network and cloud execution environments.

The publications listed below contribute to the thesis are grouped by the three parts:

Part II - Cloud RAN & Massive MIMO Resource Allocation

1. **Haorui Peng**, Emma Fitzgerald, William Tärneberg, Maria Kihl, 5G Radio Access Network Slicing in Massive MIMO Systems for Industrial Applications, *2020 Seventh International Conference on Software Defined Systems*

   Contributions: The lead author in writing the manuscript, was responsible for problem definition, system modelling, solution design, simulation implementation, experiments design and result analysis.

2. **Haorui Peng**, William Tärneberg, Emma Fitzgerald, Maria Kihl, Is Cloud RAN a Feasible Option for Industrial Communication Network?, *Journal of*

Communications Software and Systems, 2021, 17(2), pp. 97-106

Contributions: The lead author in writing the manuscript, was responsible for problem definition, system modelling, simulation implementation, experiments design and result analysis.

Part III - Cloud Control Systems in the Wild

3. **Haorui Peng**, William Tärneberg, Emma Fitzgerald, Fredrik Tufvesson, Maria Kihl, Evaluation of Control over the Edge of a Configurable Mid-band 5G Base Station, *2022 IEEE 6th International Conference on Fog and Edge Computing*

   Contributions: The lead author in writing the manuscript, deployed the 5G RBS deployment with the help of engineer from Ericsson, was responsible for system implementation, experiments and analysis.

4. **Haorui Peng**, William Tärneberg, Emma Fitzgerald, Maria Kihl, Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems, *2023 International Conference on Software, Telecommunications and Computer Networks*

   Contributions: The lead author in writing the manuscript, was responsible for Problem definition, system and experiment design, system implementation, performance analysis.

Part IV - Punctual Cloud for Time-critical Cloud Integrated Systems

5. **Haorui Peng**, William Tärneberg, Maria Kihl, Latency-aware Radio Resource Allocation over Cloud RAN for Industry 4.0, *2021 International Conference on Computer Communications and Networks*

   Contributions: The lead author in writing the manuscript, was responsible for problem definition, system modelling, solution and framework design, simulation implementation, performance analysis.

6. **Haorui Peng**, William Tärneberg, Emma Fitzgerald, Maria Kihl, Punctual Cloud: Unbinding Real-time Applications from Cloud-induced Delays, *2021 International Symposium on Networks, Computers and Communications*

   Contributions: The lead author in writing the manuscript, was responsible for

problem definition, system modelling, solution and framework design, system implementation, performance analysis.

7. **Haorui Peng**, Fatemeh Akbarian, William Tärneberg, Maria Kihl, Punctual Cloud: Achieving Punctuality for Time-Critical Cloud Control Systems, *2023 IEEE International Conference on Cloud Networking*

   Contributions: The lead author in writing the manuscript, and the sections related to control algorithm was contributed by Fatemeh Akbarian. Haorui Peng was responsible for problem definition, system modelling, framework design and system implementation. The theoretic solution is co-designed by Haorui Peng and Fatemeh Akbarian, among which the control algorithm is designed and implemented by Fatemeh Akbarian, the delay estimation algorithm is designed by Haorui Peng. The experiments and performance analysis was conducted together by Haorui Peng and Fatemeh Akbarian.

Part II

Cloud RAN & Massive MIMO Resource Allocation

Scenario Description

This part focuses on resource allocation challenges within a RAN system specifically designed for industrial settings, examining the MAC layer as a case study. Timely resource allocation is vital for any RBS, ensuring that UEs can establish radio communications promptly for data transmission.

This part unfolds in two chapters, each addressing resource allocation within different RANs. Initially, we explore a network slicing scheme for the MAC layer of a traditional RAN, where the RRH is co-located with the BBU at the RBS site (Chapter 5). Here, we assess the potential for integrating diverse types of industrial UEs into a single RAN without diminishing performance. Following this, we delve into the intricacies of Cloud RAN in the same industrial context. Our focus here is on the practicality of shifting MAC layer functionalities to a cloud environment that relies on general-purpose computing resources (Chapter 6).

To set the context, we outline a representative industrial automation scenario, as visualized in Figure 4.1. Given that the average communication range in manufacturing settings is generally limited to 200m, as noted in [HMH18], we make the assumption that all UEs fall within the coverage area of a single RBS. As a result, all UEs connect to each other via a mobile network facilitated by a one-cell Massive MIMO RBS. This necessitates the allocation of radio resources to enable seamless communication. The essential goal of the work in the following two chapters is to explore the practicality of using different RAN technologies equipped with Massive MIMO to support communications for industrial UEs.

Industrial Units

In the industrial setting under discussion, the UEs are designated as **industrial units**, comprised of an array of sensors, actuators, and controllers situated within an industrial facility. These components are spatially distributed, and their interconnected

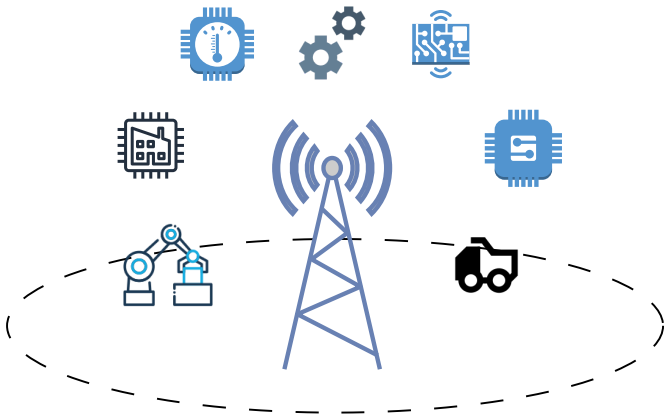


Figure 4.1: Example of an industrial scenario where all the units communicate over a single-cell radio access network.

feedback loops operate via the mobile network. Sensors serve dual functions: they monitor the states of the control plants as well as environmental parameters.

The industrial units have different requirements of QoS, and we can categorize the communication of these units into two types based on their QoS requirements and traffic patterns, URLLC and mMTC. Communications for the control loops are all categorized as URLLC type traffic, and they typically have strict QoS requirements. An industrial scenario requires for this type of traffic with network latency less than 10ms and availability within the range of 95%-99.999% [Gro19]. Meanwhile, there are also numerous devices sending out occasional monitoring messages about the plant environment. These monitoring devices are installed and distributed throughout the whole plant and require massively many connections. They are thus categorized as mMTC.

Prior to transmitting any message via the radio network, each industrial unit must request radio resources from the RBS, known as "signalling for transmission". Each such request carries a predefined deadline. If an industrial unit fails to secure radio resources before this deadline, the corresponding transmission is abandoned and marked as a loss. Industrial units involved in URLLC traffic, often referred to as Critical Units (CUs), have shorter deadlines. These units should be prioritized in the resource allocation process due to their more stringent, time-sensitive QoS needs.

Radio Resource

In our study, we focus on resource allocation problem at the MAC layer of a Massive MIMO system, which employs a 100-antenna configuration and utilizes TDD and OFDM for modulation, as detailed in Section 2.2.2. The time-frequency space of the Massive MIMO system is partitioned into **coherence blocks**, which are allocated for uplink data, downlink data, and uplink pilot transmissions. A coherence block represents the longest duration where the channel properties are considered consistent in both time and frequency domains. Uplink pilots within each coherence block serve as the resources that must be allocated to the UEs before data transmissions can begin, as these pilots enable the RBS to estimate the CSI for each UE.

In an ideal scenario, each coherence block would accommodate as many UEs as there are mutually orthogonal pilots, ensuring equal channel estimation quality for all UEs. However, real-world conditions often require deviations from this optimal setup. For example, some UEs may need multiple pilots for more precise CSI estimation, consequently reducing the overall number of UEs that can be served within a single coherence block. Alternatively, in high-density or "crowded" scenarios where the number of active UEs surpasses the available orthogonal pilots, a specialized pilot scheduling algorithm becomes essential to manage the equitable allocation of these limited resources among all active UEs.

Chapter 5

Resource Allocation in Traditional RAN

A MAC Layer Network Slicing Scheme for Two Types of Industrial Units

Before delving into the feasibility of implementing Cloud RAN in an industrial context, we first examine the traditional RAN architecture, which inherently does not introduce system-induced delays. To seamlessly integrate multiple types of industrial units into a single RAN, a synergistic approach involving both network slicing and massive MIMO—both key advancements in the 5G era—could address the limitations commonly encountered in industrial wireless networks [GLA17].

Network slicing offers the ability to customize network services over a shared physical infrastructure, tailored to meet the specific performance criteria of individual customers [GSM17]. This technology provides isolated access to the network for only those UEs that have subscribed to a particular slice. As such, network slicing emerges as a promising solution for satisfying the requirements of channel separation essential to industrial communications.

Although network slicing is predominantly implemented in the CN, where a collection of Virtual Network Functions (VNFs) are configured and interconnected through virtual networks on a programmable infrastructure [Ord+17; Bek+18], a surge of interest is evident in RAN slicing. For example, [KN17] introduces a framework for enforcing network slicing within the RAN. This work integrated a slice resource manager and resource mappers to facilitate a two-tiered scheduling process. Various studies, such as [Aij17], showcase a radio resource slicing framework using LTE-A air interfaces. Meanwhile, others like [GLK14] have proposed slicing planes for

RAN with emphasis on aspects like inter-cell interference and resource grid isolation. Notably, [Fil+22] integrated deep reinforcement learning into a Software Defined Networks (SDN)-based RAN slicing, focusing on radio resource allocation, with the overarching objective of fulfilling the QoS criteria for both URLLC and eMBB services.

Recently, there has been a heightened focus on network slicing challenges within the domain of massive MIMO, as evidenced by numerous research groups. For instance, [Yan+23] proposed a two-layered scheduler underpinned by deep reinforcement learning to facilitate network slicing in massive MIMO. Their approach splits the scheduling mechanism, with the upper layer focusing on inter-slice bandwidth allocation and the lower layer concentrating on resource block assignments to individual users within each slice. Furthermore, [Liu+23] tackled a distributed massive MIMO system, which in our perspective aligns with the paradigm of C-RAN. This study leveraged a non-orthogonal scheduling strategy, optimizing both beamforming design and RRH selection to boost the energy efficiency for URLLC and eMBB users. In this chapter, we introduce a network slicing approach at the MAC layer for a massive MIMO-based RAN. The aim is to seamlessly accommodate diverse industrial UEs within a singular cell mobile network while striving to meet the QoS demands of all UEs. To the best of our knowledge, this was one of the first studies into addressing the intricacies of network slicing specifically for massive MIMO systems at the time of publication.

5.1 Targeted System

The target system aligns with the initial framework described earlier before the chapter starts, wherein industrial units from two distinct traffic categories communicate via a single-cell RAN supported by a Massive MIMO system. The units generating URLLC traffic, termed as **Critical Units (CUs)**, are prioritized for radio resource allocation due to their time-sensitive characteristics.

Giving that there are p orthogonal uplink pilots available within a coherence block of the RBS, the maximum number of industrial units that the RBS can accommodate in a single coherence interval is constrained to be p or fewer. However, we consider a scenario where the cumulative number of connected industrial units surpasses the available pilots within a given coherence block. Consequently, a conventional approach of assigning one pilot per unit in each coherence interval becomes impractical. To address this constraint, a dynamic resource allocation strategy is essential for satisfying the QoS needs of all participating industrial units.

5.2 Network Slicing

To accommodate the resource requests of all industrial units in the given scenario, while also fulfilling the QoS requirements for different traffic categories, we employ a network slicing approach. In this scheme, different categories of industrial units are subscribed to their respective network slices. Since we identify two distinct categories of units in our scenario, these units are subscribed to two separate network slices operating atop our single-cell radio system:

1. **URLLC slice** S_1: In this slice, the units are given higher priority for pilot allocation and engage in periodic transmission requests.
2. **mMTC slice** S_2: The slice serves numerous units with aperiodic transmissions, the subscribed units have lower priority for pilot allocation.

We propose a dynamic network slicing scheme that is capable of:

1. enabling the coexistence of the units under two traffic categories with different QoS requirements;
2. isolating part of the radio resources for one high priority slice in order to meet a certain network performance;
3. preventing the high-priority units being interfered with the units of the second, low-priority, slice;
4. being generalized to more than two slices, facilitating a network slicing solution for several traffic classes with different priorities and QoS requirements.

Defining a slot time T_s to be equal to one coherence interval, hence the channel is time-invariant during this period. It is also assumed that the remaining time-frequency resources within a coherence block are adequate to complete full data transmissions for each unit that has been allocated a pilot during the slot.

The number of units subscribed to each slice is K_1 and K_2, respectively, such that $K = K_1 + K_2$. For CUs in S_1, each is allocated p_1 pilot signals, and p_1 may exceed one. This allows the RBS to obtain a more accurate CSI. In contrast, for slice S_2, we assume that each unit acquires only one pilot signal ($p_2 = 1$) in order maximize the number of served units under the category of mMTC. During each coherence interval, we consider that $\tilde{K}_1$ and $\tilde{K}_2$ units from S_1 and S_2 actively signal for transmission. Given a total number of p pilots available in a coherence block, the following condition must be met to accommodate all signalling units.

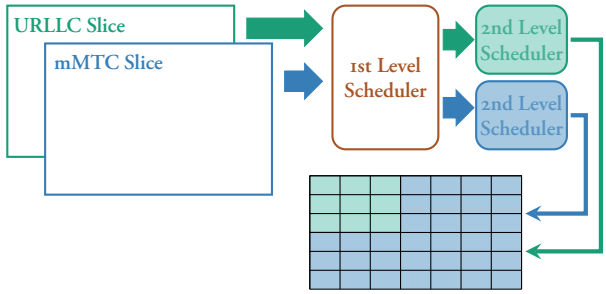


Figure 5.1: An example of the two-level MAC scheduler tailoring the time-frequency space into two slices. The first level scheduler allocates the resources to each slice, and each second level scheduler assigns the pilots to their subscribed units.

$$\tilde{K}_1 p_1 + \tilde{K}_2 p_2 \leq p. \tag{5.1}$$

The network slicing scheme we proposed employs a two-level MAC scheduler, as depicted in Figure 5.1. The first level of the MAC scheduler oversees inter-slice resource allocation. It partitions the available radio spectrum into two distinct segments, each dedicated to a specific network slice. The second level manages intra-slice scheduling, determining how the pilots allocated by the first level are distributed among the units within that slice. Units interact exclusively with their respective slices and receive pilot assignments from these second-level schedulers.

The first-level scheduler operates on a straightforward priority-based strategy, where S_1 is consistently given precedence over S_2, in order to meet the QoS requirements of CUs. The scheduler allocates the necessary number of pilots to S_1 until all available pilots have been assigned. Should there be any remaining unallocated pilots after this process, they are then designated to S_2.

Each second-level scheduler employs a specific algorithm for the intra-slice allocation of pilots to individual units. Here we will explore three commonly used scheduling algorithms:

- **First Come First Served (FCFS)**: The scheduler keeps two separate queues to manage incoming requests from the units in both slices, allocating the pilots based on the order in which requests arrive. The reliability of S_1 is assured as long as the incoming traffic does not exceed the system's maximum capacity, ensuring that no resource is wasted. However, this approach necessitates that the scheduler has full awareness of the queuing information, incurring additional computational and signalling overhead to maintain and gather this information.

- **Round Robin with partial Queuing information** (**RRQ**): In this method, the scheduler does not need to acquire full queuing information. During each coherence interval, it instead iterates through all the units subscribed in each slice and assigns pilots to the units that are signalling for transmission. In this way, only one request per unit is served during each coherence interval, but the overhead would be lower than FCFS since RRQ does not serve the request in order of arrival time.

- **Round Robin with No Queuing information** (**RRNQ**): In this method the scheduler does not acquire any queuing information; instead it assigns pilots to each unit subscribed to the slice, despite whether it is signalling for transmission or not at the moment. This yields a static scheduling approach. The scheduler groups a number of time slots into an **allocation frame** to assign the pilots. Such a frame time is equal to the deadline of each request. In this way, it is guaranteed that every transmission request will be assigned a pilot within its deadline. But it also leads to an upper bound on subscribed units to a slice, since the length allocation frame is fixed. During each allocation frame, the scheduler iterates through all the units subscribed to S_1 and assign the pilots to each of the subscribed units to serve one request from each. This does not incur any overhead for searching for active devices. However, it will result in numerous wasted pilots since some units may not be signalling for transmissions during an allocation frame. Also, this scheduling method is not applicable to S_2, since S_2 hosts an enormous number of units.

The second-level scheduler within each slice has the flexibility to employ different scheduling algorithms. Our evaluation will demonstrate that the choice of scheduling algorithm in one slice has a negligible impact on the QoS experienced in the other slice. This suggests that each slice can optimize its internal scheduling strategy without adversely affecting the performance of other slices.

5.3 Simulation

The network slicing scheme we proposed was evaluated through simulations. The full source code for our simulation program, along with the experiments and data, is publicly available on GitHub[1] for further examination and reuse. Our implementation leverages an existing massive MIMO MAC layer simulator [Stå]. The simulation is coded in Python and designed for extensibility, allowing the incorporation of various scheduling algorithms at both the first and second levels of the scheduler. Moreover, it

[1] `https://github.com/HaoruiPeng/slicing-simulator`

offers the flexibility to define custom traffic profiles for each slice. This can be done by specifying parameters such as the inter-arrival distribution, deadlines, required number of pilots, and the number of industrial units under different categories.

5.3.1 Simulation model

In our simulation framework, a set number of industrial units are subscribed to each of the two network slices. These units within each slice follow a specified arrival distribution, reflecting the expected traffic profiles for that category. Resource allocation is then carried out using our proposed two-level scheduler, which assigns the available pilots to the units in each slice based on the predetermined rules and priorities.

In the simulation, units within the URLLC slice S_1 have transmission requests that occur with a nearly constant periodicity defined by d_1, adjusted slightly by a small variance ω_1 that follows a normal distribution $\omega_1 \sim \mathcal{N}(1, 0.05)$. On the other hand, units in S_2 have a different request pattern. Their transmission requests follow a Poisson distribution with a longer average inter-arrival time d_2, indicative of aperiodic sensor data transmissions.

The deadlines for the transmission requests in both slices are set to match their respective average inter-arrival times. When a unit is allocated a pilot before the deadline expires, a connection is established between that unit and the RBS, allowing for successful data transmission within the current coherence interval. If, however, the deadline is not met, the transmission request is discarded, and the transmission is considered a failure.

The massive MIMO MAC layer simulator provided in [Stå] is an event-based simulator with basic traffic generation functions and scheduling algorithms. In the simulator, a **transmission event** is defined as the moment a unit submits a transmission request to the RBS. These events are generated for each unit based on the distribution characteristics specific to their slice, as previously described. Our extension to this simulator segregates units into distinct slices, contingent upon their traffic profiles. Additionally, it integrates our two-level scheduling scheme for pilot allocation. Each incoming request is labelled with essential metadata: the slice it subscribes to, its arrival time(the point in time the request is triggered), and its deadline (the cut-off time after which the request will be discarded if a pilot is not allocated).

A **scheduling event** is initiated every T_s to manage active transmission requests. During this event, the scheduler marks requests that have outlived their deadlines as expired. It then proceeds to allocate pilots to the remaining active requests, using our two-level scheduling framework, until all available pilots for the current time slot are

Table 5.1: Network and traffic parameters used in the simulation.

Variable name	Value	Symbol
Simulation length	10000 ms	L
Slot time	0.5 ms	T_s
Available pilot signals per slot	12	p
Mean inter-arrival time in S_1	1 ms or 10 ms	d_1
Mean inter-arrival time in S_2	50 ms	d_2
Number of pilot each device requires in S_1	1 or 3	p_1
Number of pilot each device requires in S_1	1	p_2
Traffic load in S_1	[0.1:0.1:1.1]	ρ_1
Traffic load in S_2	[0.1:0.1:1.5]	ρ_2
Number of devices in S_1	$\rho_1 p d_1 / p_1 T_{slot}$	K_1
Number of devices in S_2	$\rho_2 p d_2 / p_2 T_{slot}$	K_2
Schedulers	FCFS, RRQ or RRNQ	

exhausted. Requests that receive the requisite number of pilots for CSI estimation are designated as "served".

In S_2, each unit is allocated just one pilot ($p_2 = 1$), which suffices for achieving the minimal CSI quality essential for the ensuing transmission. Conversely, the S_1 slice, tailored for higher reliability, allows each unit to be allocated multiple pilots ($p_1 \geq 1$), thereby enhancing CSI accuracy as dictated by the specific needs of the CUs.

All unresolved requests are tagged as "pending" and rolled over to the next scheduling event for reconsideration. Importantly, transmission requests are generated independently of the scheduling events, in line with their respective statistical distributions. This leads to that units can have multiple pending transmission requests, each of which must be allocated the required number of pilots for successful service.

5.4 Evaluation

Our experiments have explored various combinations of the scheduling algorithms outlined in Section 5.2. The focus is to investigate the impact of increasing of traffic load in each slice on QoS performance. We define the traffic load, denoted by ρ_i, in slice S_i, as a function of the number of subscribed units K_i.

$$\rho_i = \frac{K_i p_i T_s}{p d_i}, \; i = 1, 2. \tag{5.2}$$

The network and traffic parameters employed in our simulation are detailed in Table 5.1. These parameters are derived from the 100-antenna test-bed of the Lund Massive MIMO (LuMaMi) system under a high-mobility scenario[Vie+14]. Our

evaluation focuses on three key aspects: the performance of the URLLC slice, the performance of the mMTC slice, and the degree of isolation between the two slices. The subsequent sections elaborate on the experimental setup for the three evaluations.

Latency and reliability performance in URLLC slice

To assess both the latency and reliability performance in the face of increasing traffic load ρ_1 in slice S_1, we held the traffic load ρ_2 in slice S_2 at a constant value of 0.5. We employed the FCFS algorithm as the second-level scheduler in S_2. For each simulation run, ρ_1 was determined according to the profile specified in Table 5.2. In this experiment, the units within S_1 demanded low-latency and moderate channel reliability. All three scheduling algorithms were implemented in S_1 for comprehensive evaluation.

Isolation between two slices

This experiment aims to investigate whether the performance of slice S_2 is affected by the choice of the second-level scheduling algorithm in the higher priority slice S_1. The S_2 performance was examined when it utilizes RRQ, while concurrently applying all the three scheduling algorithms in S_1. For each algorithm applied in S_1, we fixed the traffic load ρ_1 at a constant value of 0.5 in S_1. We then varied the traffic load ρ_2 in S_2 from 0.1 to 1.5, based on the parameters outlined in Table 5.1.

Latency and connection performance in mMTC slice

This experiment is designed to assess QoS performances in slice S_2, specifically focusing on latency as the traffic load ρ_2 in the slice increases. For the sake of this study, we maintain a constant traffic load $\rho_1 = 0.5$ in S_1. The RRNQ scheduler is employed for S_1, while both the FCFS and RRQ methods are evaluated for S_2. The traffic load ρ_2 is generated according to the parameters specified in Table 5.2. Additionally, this experiment aims to explore the upper limit units can be accommodated in S_2, using network parameters derived from the LuMaMi test-bed.

5.4.1 Performance Metrics

The evaluation employs on two key metrics to evaluate the performance of each slice under the proposed network slicing scheme: **average waiting time** and **loss rate**.

Table 5.2: Parameters used for the three evaluations. Entries with dashes are varied in the corresponding evaluation.

Slice	Variable	Eval. (1)	Eval. (2)	Eval. (3)
S_1	ρ_1	–	0.5	0.5
	p_1	1	3	1
	d_1	1 ms	10 ms	10 ms
	scheduler	–	–	RRNQ
S_2	ρ_2	0.5	–	–
	p_2	1	1	1
	d_2	50 ms	50 ms	50 ms
	scheduler	FCFS	RRQ	–

- **Average waiting time** This metric measures the duration a unit waits to receive a pilot allocation following its initial transmission request. It serves as an indicator of the slice's latency performance.
- **Loss rate**: This is calculated as the proportion of discarded requests relative to the total number of requests made by all the units during an experiment. It serves as a barometer for the slice's reliability.

5.5 Evaluation Results

This section outlines and analyses the results of our evaluation. It's worth noting that the 95% confidence intervals for average waiting times across our experiments fall within 1.6% of the respective mean values.

5.5.1 Latency and reliability performance in URLLC slice

First, we explore how varying scheduling algorithms respond to an increase in traffic load ρ_1 for slice S_1. Given that S_1 has higher priority, its performance should be independent of S_2's traffic. Theoretically, the packet loss should be 0 when load ρ_1 is less than 1 with all scheduling methods. However, as illustrated in Figure 5.2, the RRNQ scheduler does cause a loss in the transmission requests even under low traffic load. This is attributed to the variance of the inter-arrival times for the requests in S_1. In cases where two successive requests from the same unit occur within the same allocation frame due to this variance, the RRNQ scheduler would likely discard the second request.

Therefore, this loss rate is predominantly influenced by the variance in inter-arrival times, rather than the arrival period. A similar effect can be observed with the RRQ

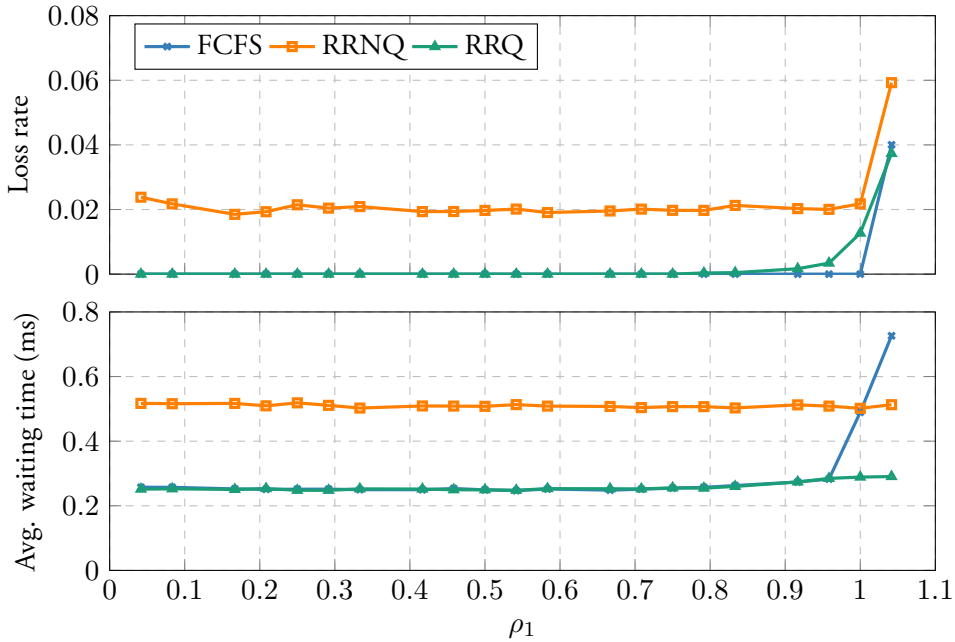


Figure 5.2: Loss rate and average waiting time performance of slice S_1 with different schedulers: The RRNQ method exhibits around a 2% packet loss even under light load conditions and has a higher average waiting time compared to other schedulers. When using FCFS, the average waiting time escalates quickly as the system nears full load.

scheduler under heavy load, particularly when deadlines are stringent and variance in inter-arrival times is high.

Figure 5.2 also reveals the RRNQ scheduler's impact on average waiting times. While the average waiting time for FCFS and RRQ schedulers equates to half a slot time, it increases to half a RRNQ allocation frame time due to the scheduler's behaviour.

When ρ_1 nears 1, FCFS's average waiting time spikes, almost hitting each request's deadline at $\rho_1 = 1.1$. This behaviour arises because FCFS is queue-dependent, and the queue length scales with the load of the slice. By contrast, in case of RRQ and RRNQ, a request's waiting time depends only on the number of pending requests from its own units.

It is worth noting that a trade-off exists between the computational and signalling cost and the performance of each scheduler. While RRNQ minimized signalling costs, it is not applicable to critical applications that require ultra-reliability. FCFS performs well when the system is not fully loaded, but it incurs higher computational and signalling costs due to queue maintenance. Furthermore, FCFS's reliability may be

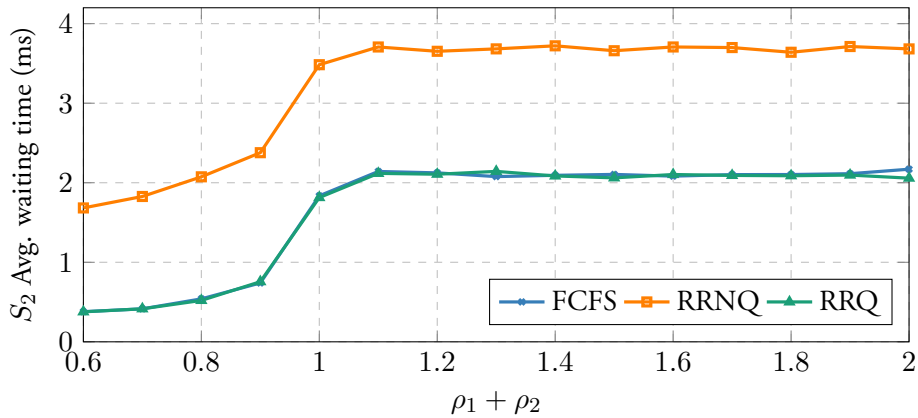


Figure 5.3: Average waiting time in S_2 based on traffic parameters specified in Eval.(2) of Table 5.2. The x-axis shows the overall system traffic load $\rho_1 + \rho_2$, with ρ_1 held constant at 0.5. The legend indicates the scheduling algorithms utilized in S_1 for each set of experiments, while S_2 consistently employs RRQ.

compromised when the system is nearing full capacity, as bursty traffic could rapidly saturate request queues.

5.5.2 Isolation between two slices

Figure 5.3 presents the impact of different scheduler choices in S_1 on the QoS performance of S_2, which reflects on the degree of isolation between S_2 and S_1. The figure illustrates the average waiting time in S_2 under three different scheduling algorithms employed in S_1. The results reveal that the performance of S_2 remains largely unaffected by the choice of S_1 scheduling algorithm when the system is not operating at full capacity. However, when RRNQ is used in S_1, S_2 experiences a longer average waiting time compared to other scenarios. This outcome aligns with our previous observation in Figure 5.2 that RRNQ results in longer average waiting times in S_1, which in turn affects S_2 due to its lower priority relative to S_1.

5.5.3 Latency and connection performance in mMTC slice

To assess the impact of different scheduling algorithms in S_2, we plotted its average waiting time as ρ_2 increases in Figure 5.4. In these experiments, S_1 consistently employs RRNQ and maintained the constant traffic load of $\rho_1 = 0.5$. We compared the performance of S_2 when it uses either FCFS or RRQ as its scheduling algorithm. Note that the RRNQ algorithm is not evaluated for S_2 since it is unfeasible for an mMTC slice with a large number of subscribing units.

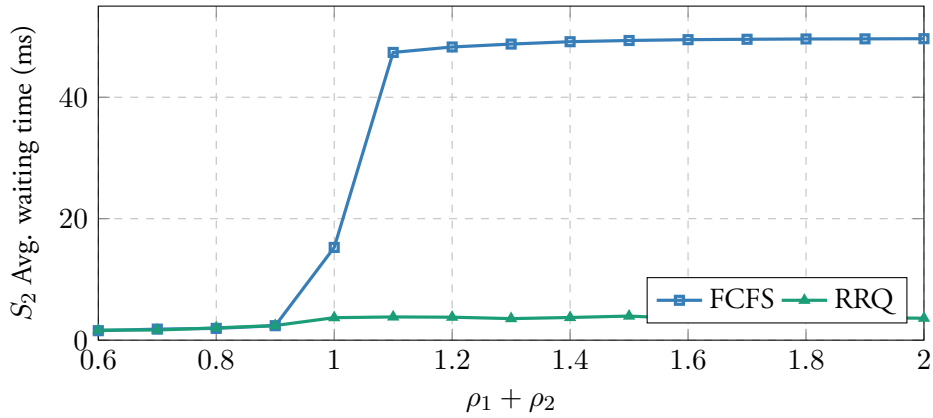


Figure 5.4: The average waiting time in S_2 under traffic parameters given in Eval.(3) of Table 5.2. The x-axis shows the overall system traffic load $\rho_1 + \rho_2$, wherein ρ_1 is constant at 0.5. The figure shows the average waiting time when S_2 applies FCFS and RRQ, along with RRNQ in S_1.

In this setup, the constant traffic load of 0.5 in S_1 results in a traffic bottleneck in S_2 beginning at $\rho_2 = 0.5$ – the point at which the system becomes fully loaded. The behaviour of the average waiting time using FCFS in S_2 parallels that observed in S_1 (Figure 5.2). This is because the waiting time under the FCFS algorithm is highly sensitive to the overall system load, and it starts to escalate substantially when the system is overloaded. Conversely, the average waiting time under RRQ remains stable even when the system is overloaded. This is due to the system blocking the incoming units and discarding their requests when overloaded. The average waiting time performance remains equivalent to conditions where $\rho_1 + \rho_2 = 1$, as only successfully served requests are factored into the calculation.

Figure 5.5 plots loss rate against the number of connections in S_2, providing corresponding overall system traffic load at the top of the figure. The experiment leverage the specified traffic profiles described in Eval.(3) of Table 5.2 and network parameters from Table 5.1. Given that S_1 operates at a constant traffic load, the loss rate in S_2 is primarily dependent on its own traffic load. As such, both the FCFS and RRQ scheduling algorithms display similar behaviour. The results indicate that the system can accommodate up to 600 mMTC units in S_2, without any packet loss, irrespective of the scheduler used.

Our simulation results revel that the performance of S_2 is principally affected by the traffic loads in both S_1 and S_2, as well as by the choice of scheduling algorithm in S_2, while the scheduling algorithm used in S_1 has relatively minor impact on S_2's performance. Given that S_2 does not necessitate ultra-reliable service, a certain degree of packet loss could be tolerated to facilitate a larger number of concurrent

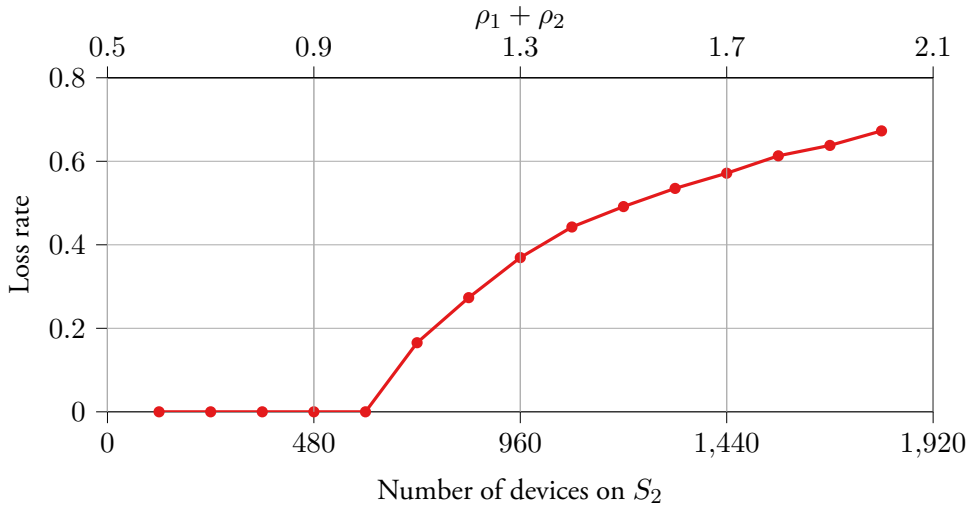


Figure 5.5: Loss rate in S_2 as a function of number of connections. The graph demonstrates that, under the given network parameters, the system can accommodate up to 600 units in S_2 without incurring any loss. This capacity threshold is equivalent to maintaining an overall system traffic load of $\rho_1 + \rho_2 = 1$.

connections in S_2.

5.6 Conclusion on the Network Slicing Scheme

In summary, this chapter presents a MAC layer resource allocation approach utilizing the network slicing concept for a single-cell massive MIMO RAN system, aiming to serve multiple types of UEs within an industrial context. The study employs a two-level MAC scheduler and evaluates three different algorithms—FCFS, RRQ, and RRNQ—each with distinct merits and limitations.

The results affirm the system's capacity to support diverse UEs and fulfil their QoS requirements via the proposed slicing scheme. However, the selection of the scheduling algorithm needs to be fine-tuned according to specific use-cases and traffic patterns. Among the tested algorithms, RRNQ offers the least overhead but falls short in ensuring ultra-reliability, making it unsuitable for scenarios requiring a massive number of connections. Conversely, FCFS offers higher reliability but may incur significant overhead, particularly when the system reaches full capacity. RRQ offers a balanced approach, maintaining consistent waiting times even under high loads but at the expense of increased signalling overhead.

The study serves as an initial investigation into the viability of deploying a traditional single-cell RAN system in industrial settings, achieving multi-tenancy and varying QoS requirements. The next chapter will extend this research to a Cloud RAN system to explore its feasibility and performance in meeting the QoS needs of industrial units, particularly CUs under URLLC traffic category.

Chapter 6

Resource Allocation in Cloud RAN

Addressing the Needs of Critical Industrial Units

In this subsequent chapter, we extend our investigation into resource allocation within industrial settings, shifting our focus from traditional RAN systems to Cloud RAN systems. Building on the earlier validation of traditional RAN deployment in industrial scenarios, we now aim to assess the viability of implementing Cloud RAN with similar conditions, taking into account the distinct delay characteristics inherent to the Cloud RAN architecture, which must be carefully considered.

6.1 Targeted System

In alignment with the focus of the previous chapter, this chapter also explores an industrial scenario. However, the communication here is facilitated through a Cloud RAN mobile network. This Cloud RAN system employs the same Massive MIMO antenna system, LuMaMi in Chapter 5, but with a notable distinction: the BBU is located in a cloud-based, general-purpose data centre, forming a BBU pool. This pool is then interconnected with the RRH via Ethernet fronthaul link.

The system operates based on 3GPP function split option 6, which divides the BBU's functionalities between the MAC and PHY layers. In this configuration, all PHY layer functionalities are executed at the RRH, eliminating the need for transmitting raw baseband data blocks over the Ethernet fronthaul link to the cloud-based BBU pool. A schematic representation of this industrial scenario, wherein industrial units communicate via the Cloud RAN system, can be found in Figure 6.1.

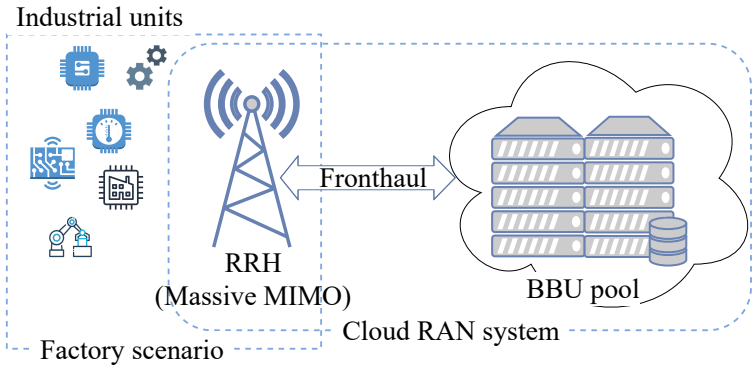


Figure 6.1: An industrial scenario where all industrial units communicate over Cloud RAN

In this chapter, our focus narrows to the resource allocation for a specific type of industrial unit, whose QoS is particularly sensitive to the latency challenges introduced by Cloud RAN architectures. These units, categorized under URLLC in the previous chapter, are referred to as **Critical Units (CUs)**. They necessitate rapid resource allocation to facilitate their frequent, periodic transmissions.

The migration to Cloud RAN for industrial communication poses latency challenges specifically for CUs, owing both to network factors and the cloud-based execution environment of the BBU pool. In contrast, the other type of units, categorized as mMTC and considered in Chapter 5, are designated as non-CUs. Their communication serves as background traffic in this context, and their QoS will not be the focus of this section as they do not have stringent latency requirements.

6.2 Delays in a Cloud RAN System

Transition to cloud-based resource allocation approach poses larger challenges compared to traditional RAN systems. This complexity arises from the additional abstraction layers introduced in the traffic pathway between the RRH and the BBU pool. As a result, by the time the scheduling decisions, made by the BBU pool, reach the RRH, they might no longer be appropriate for the current state of the CUs. This misalignment is primarily due to the lag in information exchange caused by inherent delays. The objective of this chapter is to assess the feasibility of Cloud RAN deployment in light of these delay challenges stemming from both the network and the cloud execution environment.

Figure 6.2 offers an illustrative representation of the kind of delays one might expect in a cloud environment. This is our assumed delay model for the Cloud RAN system

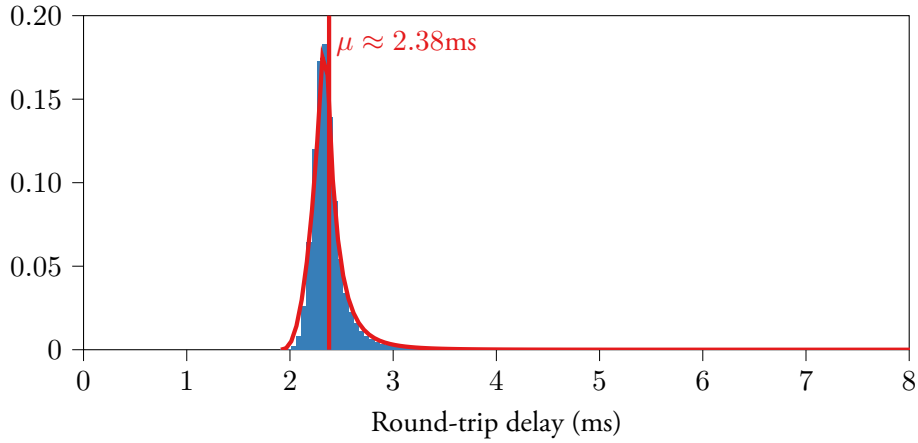


Figure 6.2: The histogram of the UDP round-trip delay measurements. The red curve is the probability density function and the mean value fitted from the histogram.

under discussion. The figure presents a histogram of round-trip delay measurements corresponding to the Docker application shown in Figure 3.2(b). As previously mentioned, this application is hosted in the ERDC in Lund, located approximately 2km (walking distance) from the massive MIMO RRH LuMaMi.

The histogram aligns with a Log-Laplace distribution, characterized by a mean value of $\mu \approx 2.38$ms. It's worth noting that this long-tailed distribution arises not only from the distance between the RRH and the BBU pool but also from the cloud execution environment within the data centre.

6.3 System and Simulation Model

In this section, we outline the simulation model for the industrial scenario depicted in Figure 6.3. As previously mentioned, the RRH is represented by the Massive MIMO antenna system LuMaMi, and the BBU pool is situated within the ERDC in Lund. The RRH and the BBU pool interact via a fronthaul link to allocate radio resources to K CUs within the coverage area of the RRH.

6.3.1 RRH and BBU Pool Model

In this simulation model, we make the assumption that a single uplink pilot of each CU is sufficient for the base station to estimate its CSI, facilitating the servicing of transmission requests within a coherence interval.

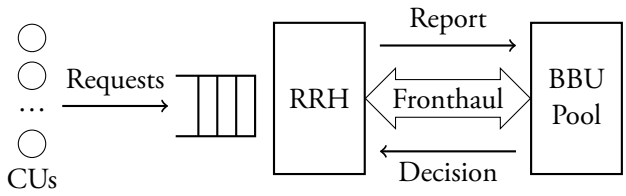


Figure 6.3: Simulation model of the system, where CUs are making transmission requests to the Cloud RAN for radio resource assignment.

In addition to the radio resource definitions outlined in Chapter 5, we introduce the factor of CU mobility, a key determinant of the coherence interval in the radio network. Under scenarios where CUs are stationary, the communication channel to the RBS experiences fewer variances compared to situations with high-mobility CUs. This results in a longer coherence interval for the radio system.

Given this, we assume that the quantity of available pilots within a coherence interval directly correlates with the interval's duration, itself determined by the CU's mobility. In massive MIMO systems, CUs are multiplexed in the spatial domain. Consequently, once a pilot is designated to a CU, the volume of its accommodated transmissions also scales with the interval's length.

We denote the smallest interval length of our system as T_c. During this time, p pilots are available. This suggests that, at most, p CUs can be allocated pilots within T_c, with each CU being served a single transmission request if assigned a pilot. We also define T_s as the actual of a coherence interval of the system, also representing the allocation timeslot in our scheduling problem, with P pilots available during each slot.

In a high mobility scenario, where the coherence interval is minimized, we have $T_s = T_c$ and $P = p$. However, as the CU's movement speed decreases, both the coherence interval T_s and the available pilots P increase proportionately.

The RRH maintains an ingress queue comprising all active transmission requests from CUs. The BBU is equipped to monitor this queue's status. With every update to the queue, the BBU dispatches a new scheduling decision, enabling the RRH to apply the updated allocation guidelines to the active CUs.

6.3.2 Fronthaul and Latency Model

The fronthaul link introduces a delay for every message transmitted over it. This round-trip delay is characterized as the time span from when a report is sent until the associated decision reaches the RRH, including the computing time in the BBU for making the decision. As previously highlighted, this round-trip delay is modelled using a Log-Laplace distribution with a mean value of μ milliseconds.

6.3.3 Traffic Model

Each CU_k, where $k \in \{1, 2, ..., K\}$, dispatched transmission requests at an average rate of λ_k. Our model takes into account the industry and IoT source level traffic models detailed in [HMH18]. We use the **homogeneous periodic traffic** as the arrival process to generate transmission requests. By following this arrival process, each CU sends requests at intervals close to a constant value of c, albeit subject to slight variations introduced by a normal distribution, implying that the average request rate for CU_k is $1/c$. Every request encompasses:

- The ID k of the CU, indicating it as a request initiate by CU_k.
- The sequence number γ, marking it the γth request by CU_k.
- A deadline D_k^γ. The deadline length for CU_k is drawn from a uniform distribution that spans from c to D, where D represents the maximum deadline length for all the CUs.

Consequently, the aggregate average request rate for the system equates to $\lambda = K/c$. The offered load to the system is contingent solely upon the count of active CUs, denoted as K, within the scenario.

6.3.4 The Scheduling Policy

At the onset of each coherence interval, the RRH forwards the details of all the active requests within the ingress queue to the BBU pool. Within our model, we term this transmitted information as the **report**. This report encapsulates both the CU ID and the deadline, represented as (k, D_k^γ), for every active request queued.

Upon receipt of a fresh report, the BBU pool analyses the data pertaining to active requests in the queue and formulates an appropriate **decision**. This decision is represented as a subset of CU IDs, denoted as $\mathcal{K} \subseteq \{1, 2, 3, ...K\}$, indicating which CUs will be allocated the pilots.

If the total count of CUs having active transmission requests within the ingress queue is fewer than the available pilots P, then all the CUs in the queue are granted a pilot. Conversely, if the number of CUs surpasses P, the Earliest Deadline First (EDF) algorithm is employed. Using this method, pilots are delegated to the P CUs with requests that have the most imminent deadlines.

6.3.5 Performance Metrics

Two performance metrics are utilized in the evaluation for the Cloud RAN system: **loss rate** and **pilot utilization**.

The pilot utilization for a given time slot j is determined by how effectively the pilots are assigned to the CUs in set $\mathcal{K}_j$. Let $\hat{P}_j$ be the decided number of pilots to be assigned to the CUs waiting in the queue. It's given that $\hat{P}_j \leq P$, and the length of set $\mathcal{K}_j$ is equal to $\hat{P}_j$, where $\mathcal{K}_j$ represents the subset of CUs being considered for assignment in time slot j.

For each CU in $\mathcal{K}_j$, the potential number of transmission requests that can be served is T_s/T_c – this is because the number of served requests is proportional to the coherence interval length. If $N_{k,j}$ denotes the actual number of active requests for CU_k in the queue during the time slot j, the number of pilots wasted, $W_{k,j}$, for CU_k is

$$W_{k,j} = \begin{cases} 0 & \text{if } N_{k,j} \geq T_s/T_c \\ \frac{T_s/T_c - N_{k,j}}{T_s/T_c} & \text{if } N_{k,j} < T_s/T_c \end{cases} \tag{6.1}$$

To determine the overall pilot utilization U_j, for the time slot j, compute:

$$U_j = 1 - \frac{\sum^{\forall k \in \mathcal{K}_j} W_{k,j}}{\hat{P}_j T_s/T_c} \tag{6.2}$$

Here, P is the total number of available pilots. A higher U_j indicates effective pilot utilization in that time slot, while a lower U_j suggests inefficiencies. By evaluating U_j over a range of time slot of total length T, we can gauge the overall resource allocation efficiency of the Cloud RAN system during the whole service period:

$$U = 1 - \frac{\sum_{j=1}^{T/T_s} \sum^{\forall k \in \mathcal{K}_j} W_{k,j}}{\sum_{j=1}^{T/T_s} \hat{P}_j T_s/T_c} \tag{6.3}$$

Given that $S_{k,j}$ represents the actual number of requests from CU_k that are served in time slot j, the loss rate of the system during T is calculated as:

$$\bar{L} = 1 - \frac{\sum_{j=1}^{T/T_s} \sum^{\forall k \in \mathcal{K}_j} S_{k,j}}{\sum_{k=1}^{K} \lambda_k T} \text{where } S_{k,j} = \min(T_s/T_c, N_{k,j}) \tag{6.4}$$

Table 6.1: Arrival Process Parameters for the Evaluation on Tolerable Round-trip Delay.

Parameter name	Value	Symbol
Arrival interval	10 ms	c_k
Number of CUs	20	K
Deadline length bounds	$\{5, 6, 8, 10, 12, 15\}$ ms	D

Table 6.2: Parameters Related to Different Mobility Scenarios in the Simulation.

Mobility scenario	High	Medium	Low
Coherence interval length T_s	0.5ms	1ms	1.5ms
Available pilots per interval P	12	24	36

We represent this calculated loss rate as $\bar{L}$ due to its derivation from the mean arrival rate λ_k of each CU. In our simulation experiments, we measured the actual number of transmission requests within the system to derive the precise loss rate, denoted as L.

6.4 Evaluation

This section details the experiment setup and the parameter configurations used in our simulations, aiming investigate the feasibility of a Cloud RAN deployment in an industrial automation scenario. The simulation is implemented with SimPy[1]. Each experiment simulated a system runtime of $T = 200\,000$ms, with 20 runs conducted for each parameter combination.

6.4.1 Latency

In our simulation, round-trip delays are generated from the Log-Laplace distribution, as illustrated in Figure 6.2, which is empirically modelled from our UDP measurements to ERDC. While the mean, μ, of the round-trip delay ranges from 0.5ms to 15ms, all other parameters of the distribution remain consistent across experiments.

6.4.2 Loss Rate

To determine if the Cloud RAN system meets the stringent requirements established by industrial standards, we've set an upper limit for the acceptable transmission request loss rate at 5%.

[1] `https://simpy.readthedocs.io/en/latest/`

The loss rate is intrinsically linked to the CUs' tolerance for their waiting time in obtaining a radio resource. With this in mind, we conducted experiments to pinpoint the maximum round-trip delay that CUs can endure, particularly when they are subject to varying deadlines. The parameters associated with the CUs arrival process are detailed in Table 6.1. For this evaluation, we selected a medium mobility scenario, with the specific parameters for this scenario found in Table 6.2.

Furthermore, to assess the maximum capacity of CUs the system can handle across different mobility scenarios, we performed experiments where all CUs maintained deadline lengths equivalent to their arrival intervals, as presented in Table 6.1. We established the round-trip delay for this evaluation at 3ms, slightly exceeding our earlier UDP round-trip measurements.

6.4.3 Pilot Utilization

When the loss requirement is satisfied, pilot utilization becomes a key performance indicator. While it is evident that a reduced loss rate can be achieved by allocating extra resources to the CUs, this strategy can inadvertently starve the background traffic, which is typically of lower priority than CU traffic, due to inefficient pilot utilization.

Given this, it is crucial to assess pilot utilization in scenarios where the loss rate is already low. In such cases, deadline lengths have minimal influence on utilization. Instead, the length of the coherence interval, synonymous with the mobility of the CUs, becomes the primary determinant. To study this, we conducted experiments across various mobility scenarios, keeping the parameters for the CUs arrival processes consistent with those in Table 6.1. The only exception was the deadline length, which was capped at 15ms for this evaluation. We further established the maximum round-trip delay at 8ms, a threshold at which discarded requests in the system are rare given this deadline.

6.5 Evaluation Results

In this section, we present the simulation results, focusing on the two primary performance metrics: **loss rate** and **pilot utilization**, as outlined in our evaluation setup.

6.5.1 Loss

In Figure 6.4, we illustrate the upper limit of round-trip delay the system can tolerate while maintaining a loss rate below 5% given the arrival processes of the CUs as

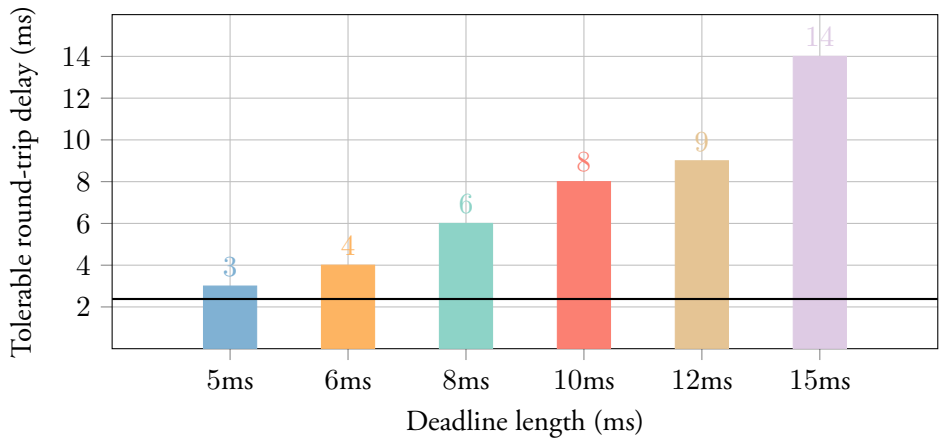


Figure 6.4: The tolerable round-trip delay with varying CU deadline lengths. There are in total 20 CUs, all with medium mobility. The dashed line indicates the mean round-trip delay from our earlier UDP measurements shown in Figure 6.2.

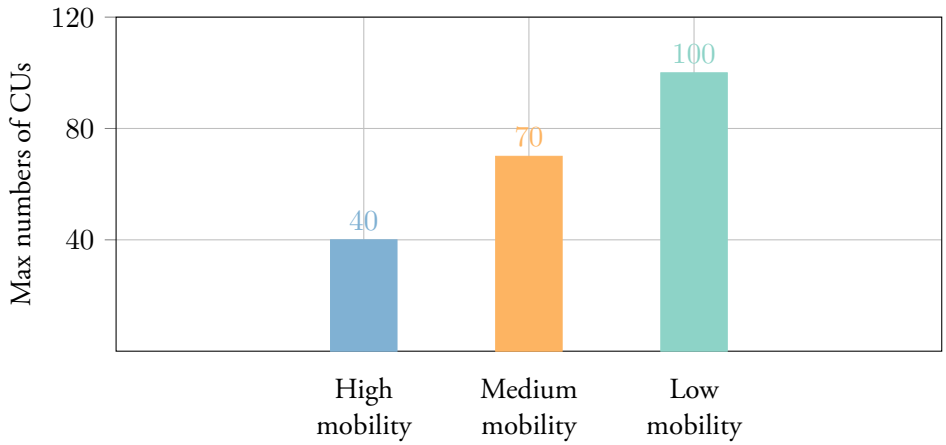


Figure 6.5: Maximum number of CUs the system can accommodate within permissible loss threshold of 5% under different mobility scenarios. Each CU has a deadline length of 10ms and the system round-trip delay is 3ms.

detailed in Table 6.1. It is evident from the figure that the acceptable delay consistently ranges from 1 to 3ms less than the associated deadline length. Therefore, if each CU's deadline mirrors its period, the inherent round-trip delay of the Cloud RAN system should not exceed the CU's transmission interval.

On the other hand, Figure 6.5 showcases the system's capability in terms of the maximum number of CUs it can serve, staying within the permissible loss limit of 5%.

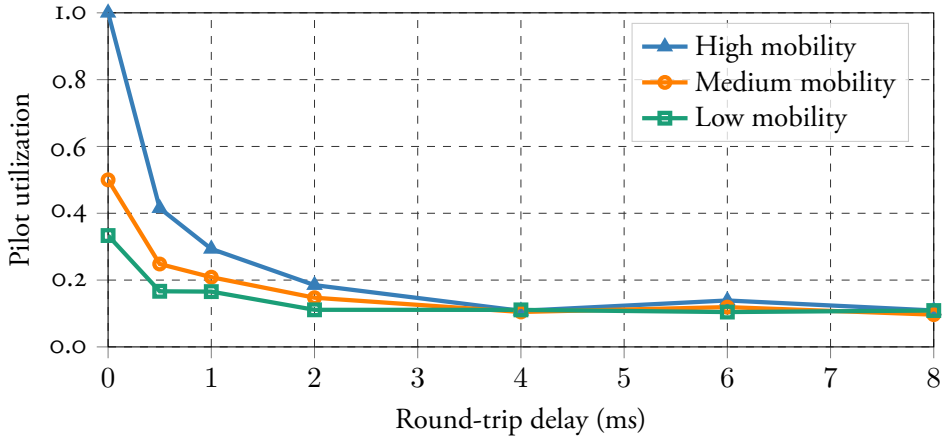


Figure 6.6: The pilot utilization when the number of CUs is $K = 20$ and each CU has a deadline length between 10 and 15ms.

This is under the assumption that all CUs have a deadline length of 10ms and experience a round-trip delay of 3ms. As anticipated, a lower mobility allows the system to accommodate more units, as the coherence interval extends. In essence, in conditions of reduced mobility, the system can manage a larger traffic load from the CUs without incurring significant losses, as opposed to scenarios with higher mobility.

6.5.2 Pilot Utilization

Figure 6.6 illustrates the relationship between pilot utilization, CU mobility, and system delay. In a system with no delay, a short coherence interval achieves full pilot utilization. However, as the interval lengthens, there is a dramatic decline in pilot utilization, dropping to a mere 40% even with a slight 0.5ms round-trip delay. This is attributed to shorter allocation slots facilitating more frequent decisions, thus allowing the system to better adapt to the dynamic ingress queue. Conversely, longer intervals entail more reserved time-frequency space for the same set of CUs during each slot. Given that the CUs' transmission intervals typically exceed the length of an allocation slot, this often results in redundant allocations, especially when the number of queued requests is less than the system's serving capacity.

As the round-trip delay between the RRH and BBU pool grows, potentially leading to outdated queue reports, pilot utilization converges to around 10%. In such scenarios, the length of the coherence intervals becomes less influential. The latency-induced misreporting frequently results in faulty allocations. As a result, a CU may be assigned a pilot based on the most recent decision, disregarding the fact that its requests have

already been addressed by prior decisions.

6.6 Conclusion on Cloud RAN under Industrial Scenario

In this chapter, we delved into a pilot scheduling process tailored for industrial Critical Units (CUs) with strict deadline requirements. While the scheduling process resides in the BBU pool, the task of allocating pilots to CUs falls to the RRH. We focus on two critical performance metrics: **loss rate** and **pilot utilization**. Employing a straightforward EDF scheduling strategy, we assessed the Cloud RAN system's ability to manage delays inherent to the resource allocation functions. Simulations were conducted to analyse the system's performance across varied scenarios.

Our experimental findings underscore the viability of deploying a Cloud RAN system in industrial automation contexts. Specifically, CUs can tolerate round-trip delays that are up to 3ms shorter than their individual deadlines. For RRHs equipped with massive MIMO capabilities, UEs exhibiting lower mobility are advantageous. They promote extended coherence intervals and subsequently diminish loss rates. In settings where unit mobility is limited, the system can effectively serve a larger group of CUs simultaneously. However, when delays are combined with extended coherence intervals, there is a notable increase in resource wastage. Such inefficiencies might limit the resources available for background traffic, which could compromise the system's overall performance.

Summary

This part delves into the MAC layer resource allocation problem within Massive MIMO tailored for industrial settings. Distinctively, two RAN architectures are dissected: traditional RBS-based RAN and the Cloud RAN. Our aim is to investigate the potential of these systems in accommodating diverse industrial units within the Massive MIMO RAN framework and assess their reliability in accommodating critical industrial entities.

The opening chapter sheds light on a resource slicing scheme within the conventional RBS framework. Herein, a two-level MAC scheduler is introduced, adeptly partitioning radio resources to serve different industrial unit categories. Three different scheduling algorithms are applied in the slicing scheme: FCFS, RRQ, and RRNQ. Simulations are conducted to evaluate system performance, ensuring compliance with the unique QoS requisites of each slice. Furthermore, the system exhibits extendability, allowing for the addition of further slices to accommodate an even broader spectrum of traffic types.

Transitioning to the subsequent chapter, the spotlight is cast on the compelling idea of integrating a cloud execution environment into the RAN architecture. The Cloud RAN system comprises a BBU pool, grounded in the GPP cloud environment. This configuration, though promising, is not without its challenges, particularly latency concerns, when functionalities span across the system. Based on the characteristics empirically modelled from cloud application response time measurements, simulations are conducted to validate the practicality of such a deployment under stringent industrial benchmarks. The findings from these evaluations are optimistic, attesting to the viability of the system in industrial automation contexts. However, a note of caution emerges concerning the potential for significant resource wastage due to inherent delays, potentially jeopardizing the resource availability for other concurrent traffic served by the system.

Part III

Cloud Control Systems in the Wild

Scenario Description

In this part of the thesis, we shift our focus to another type of cloud integrated systems the Cloud Control Systems (CCSs). CCSs represent a subset of cyber-physical systems where the controller is deployed on public or edge clouds. This controller interacts with the control plant via a network connection

When implementing controllers within the cloud, adaptations to align with cloud deployment paradigms are essential. For instance, the controller might function like a web service, responding to HTTP requests from the plant within the cyber-physical system. Consequently, achieving low network latency is essential to mirror the performance of locally controlled systems. However, in the realm of CCSs, the concerns are not limited to just the lower layers like PHY and MAC of the communication stack. As we'll discover during our evaluation of CCSs in this thesis, higher layers, including the transport and application layers, also significantly influence the network performances of CCSs.

We begin by detailing a comprehensive deployment of CCS over a 5G network, highlighting the combined effects of network infrastructure and software deployment on the control performance of CCSs. Subsequently, we evaluate the influence of various transport layer protocols on the response time of CCSs.

Model of CCS

A traditional control system comprises a controller and the corresponding plant it manages. In contrast, a cloud control system, illustrated in Figure 6.7, while retaining the same core components, differentiates by situating them apart through a network. The controller functions as a cloud service, residing in an edge or centralized cloud. The control process cycles at an optimal frequency. During each cycle, the plant communicates its current state to the cloud controller. The controller then processes this data to generate a control signal which it promptly returns. This control signal is

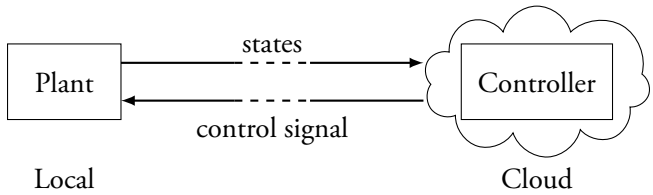


Figure 6.7: System model of a Cloud Control System

subsequently actuated by the plant.

Communication in CCS

In a CCS, where controllers are deployed as cloud applications, full-stack network protocol implementation is essential for communication between the plant and its cloud controller.

Lower-level protocols, such as PHY, MAC, and the Network layer, are inherently tied to the selected communication medium and the properties of IP networking. In our work, we have integrated various networks like 5G, Ethernet, and the Internet into our CCS designs, adapting them to different scenarios to meet distinct experimental aims

Conversely, the selection of higher layer protocols and communication frameworks rests with developers and is guided by specific use-case requirements. For our CCS design, we predominantly utilized the REST architecture, reflecting its widespread adoption in contemporary network applications. Within this setup, the plant of a CCS operates as a client, while the cloud controller serves in a server capacity. As depicted in Figure 6.7, plant states are communicated via HTTP POST requests, with the subsequent control signals being relayed as HTTP responses to those requests.

Chapter 7

A 5G-assited Cloud Control System

In this chapter, we delve into the practical deployment of a CCS tailored for time-sensitive applications. Currently, the study of mobile edge application design and validation predominantly leans on emulated or simulated environments. The scarcity of access to a comprehensive and modifiable 5G RBS has limited the scope of academic inquiry, hindering the identification of potential challenges within real-world 5G systems. The nuances of actual 5G infrastructures also pose challenges for academics venturing into application design.

Given this backdrop, there is a compelling need to offer researchers a doorway into the edge infrastructures of 5G, enabling them to experiment and refine their designs for essential applications. This chapter embarks on a journey to explore the latency characteristics at the application-level of a tunable mid-band 5G SA setup and spotlights a CCS that operates at the 5G network's edge. This system encapsulates a control loop with an integrated plant and controller mechanism. While the plant adopts the role of a client, the controller emerges as an edge service, which is co-located within the same computing infrastructure of the 5G core. The communication bridge between the client and the controller service is forged using HTTP, a protocol ubiquitously supported by modern web and cloud platforms. Our primary performance metric is the application's end-to-end response time, highlighting the viability of deploying mission-critical applications on the edge of the 5G network

A significant portion of our investigation is earmarked for the influence on the system performance by network traffic, along with radio parameters like Discontinuous Reception (DRX) timers, which play a pivotal role in dictating both the power usage and communication latency intrinsic to IoT applications.

Figure 7.1 provides a detailed illustration of the system we designed to evaluate the

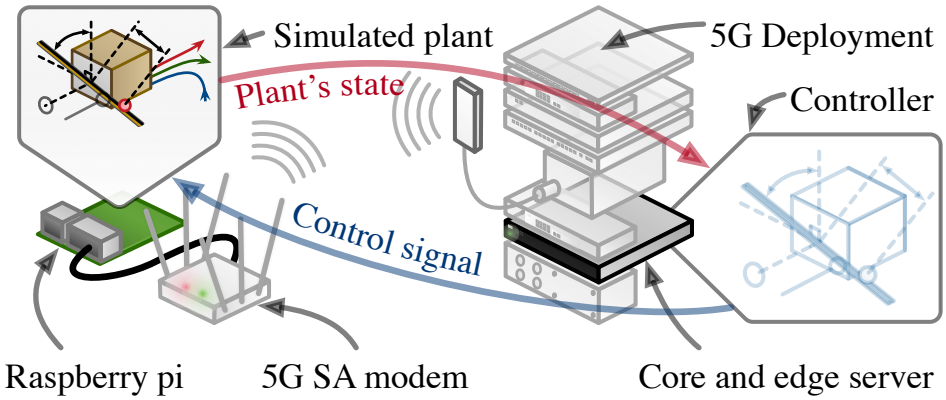


Figure 7.1: Overview of the system set-up.

performance of CCS augmented by 5G technology. This setup integrates the initial stages of the LTH 5G RBS (Figure 2.6), serving as both the communication medium and the hub for edge computing resources. Additionally, a UE is incorporated, primarily for running the plant within the CCS system.

7.0.1 System Setup

The demonstrated system is structured around two principal agents, the **client** and the **service**.

The client operates as a UE within the 5G network, utilizing a Raspberry Pi 400 with the Raspberry Pi OS, a lightweight Debian-based system. Given that the Raspberry Pi lacks embedded 5G connectivity, a WNC SKM-5xE 5G mobile modem is integrated via the device's Ethernet network interface. Operating in bridge mode, the modem facilitates a streamlined connectivity process wherein the Raspberry Pi is allocated an IP address directly from the 5G CN.

The service is an MPC controller implemented in Python by [ST]. The controller resides in a Docker container, hosted on an Ubuntu machine that concurrently serves the 5G CN. This strategic locational choice ensures that the controller resides at the 5G network's edge, a positioning that inherently minimizes transmission and processing delays. Consequently, the system's response delay becomes a product of three variables: the intrinsic latency of the 5G network, the execution time of the application, and the time investment required for processing higher-layer protocols.

In conclusion, this setup epitomizes efficiency, with every element strategically designed and positioned to enhance response times and overall performance in real-time

applications. It harmoniously integrates the advantages of cutting-edge hardware with strategic placement to deliver optimal results.

7.1 Experiment setup

To assess the efficacy of a mission-critical control process operating at the edge of a 5G network, we examine the performance at the application layer between the client and the service in our deployment. As highlighted previously, the system employs HTTP as its application protocol. The metric of focus is the **response delay** for the control signals, synonymous with the HTTP response time measured in milliseconds. This emphasis is based on:

1. Compared to mere radio/core network benchmarking, the end-to-end communication delay stands out as a central concern in CCSs. This delay aggregates both the software's processing time and the inherent network delays, both of which shape an application's overall performance.
2. The impacts from the application layer on end-to-end delays are significant in numerous real-time processes. In some cases, the computing time of an intensive process can account for a substantial fraction of the response delay. This is especially true in edge cloud environments where computing resources might be constrained relative to centralized cloud architectures.

The evaluation of application performance is three-fold:

1. Measuring the application response delay over 5G with variable application sampling rates and payload sizes.
2. Gauging the application response delay over 5G and assessing the influence of the DRX ON duration timer.
3. Evaluating the operational performance of a control process routed through 5G.

7.1.1 Performance of HTTP application running over 5G

We assess the response delay through the lens of a basic HTTP application, where the client and service interface via the early stage LTH 5G RBS as outlined in Section 2.2.3. We record the response delay at the application layer, capturing the timing at which

Table 7.1: DRX parameters

Parameter name	Value
DRX long cycle	160ms
DRX inactivity timer	100ms
DRX on-duration timer	50ms

each HTTP response is received by the client process. Our analysis is based on data derived from 50000 requests in each experiment.

To mitigate the influence of service execution time on response delay, the initial two experiments employ a straightforward HTTP "PING" application. In this setup, the client dispatches an HTTP request to the service at specified payload sizes and frequencies, and the service responses with the same message that contains in the request, ensuring minimal execution time at the service end.

When delving into network analysis, it becomes apparent that both request frequency and payload size significantly impact the data rate essential for throughput evaluation. However, in our study, we extend our focus beyond network throughput, given that the frequency is inherently tied to the control process's dynamics. Evaluating application performance under varying frequencies and payload sizes is instrumental to encompass a diverse array of application types. Consequently, our examination spans system trials with request frequencies ranging from 200Hz down to 10Hz, encapsulating control processes with sampling rates varying between 5ms and 100ms. In the subsequent sections, performance evaluations will be conducted with request intervals fluctuating within the 5ms to 100ms spectrum. Payload sizes for each packet are examined within the confines of 64 bytes and 2048 bytes.

7.1.2 Performance of application response delay affected by DRX

In our experiments, we have allocated particular attention to the DRX timers, a set of configurable radio parameters in our 5G deployment. DRX plays a major role in influencing network latency in telecommunication systems. The incorporation of DRX is pivotal in contemporary LTE and 5G systems, as outlined in [Mor21]. This significance is accentuated given the heightened emphasis on UE power consumption, especially in the era of cellular IoT.

The DRX mechanism orchestrates the operational states of UE transceivers through the manipulation of on-duration timers, inactivity timers, and DRX cycles. This orchestration facilitates enhanced battery conservation as the UE transitions to idle mode amidst periods of transmission inactivity[BI09]. However, it is crucial to acknowledge that network delay within a DRX-enabled architecture is intrinsically tied

to packet transmission inter-arrivals [BI09]. This correlation underscores the implication that control processes operating over a DRX-enabled network might experience diminished DRX impact at heightened request frequencies. Thus, the calibration of DRX parameters necessitates a tailored approach, contingent on the specific applications active over the network.

In our exploration, we extended our evaluation to encompass the performance metrics of an HTTP application operating with DRX enabled. We have delineated the specific DRX parameters integral to our evaluation in Table 7.1. Given the dependency of DRX delays on request intervals and DRX timers, our approach mirrored the methodology outlined in Section 7.1.1, where we engaged an HTTP application with request intervals from 5ms to 100ms. We maintained the payload size at a consistent 1024 bytes, aligning closely with the payload size of one of our edge-deployed control processes.

Our analysis compares the application response delay within the DRX enabled environment against a DRX disabled setting, under the same set of application parameters. This analysis is aimed at understanding the nuanced impacts of DRX on processes characterized by high-frequency operations.

7.1.3 Performance of a control process running over 5G

We broaden our assessment to include the performance metrics of a control process operating over the 5G network and edge, specifically with DRX disabled. We transition from the elementary HTTP "PING" application to deploy an emulated Ball-and-Beam (BnB) control process. This deployment aims to examine the feasibility of executing dynamical processed over a 5G network and edge computing infrastructure.

The BnB is a classic dynamic control process, where the objective is to maintain a ball balanced on a rotatable beam. For this evaluation, we set a consistent sampling rate at 50ms for the BnB process, ensuring regular monitoring of the ball's position and the beam's rotation. The payload size for each request oscillates between 700 and 900 bytes. Similar to the earlier "PING" process, the BnB plant emulation is conducted at the client end, while the controller are housed as a service in close proximity to the 5G CN, responding to HTTP requests.

Our analysis delves into the performance metrics of this CCS, benchmarking it against a parallel system where the control plant at the client end communicates with a service hosted on a machine, connected to the UE through a wired network. This comparative analysis is pivotal in understanding nuanced operational efficiencies and potential bottlenecks, offering insights into the scalability and reliability of deploying intricate

control processes over evolving 5G networks.

7.2 Performance outcomes

In this section we present the outcomes of the performance evaluation experiments described in Section 7.1.

7.2.1 Response delay vs. request intervals and payload size

A close inspection of Figure 7.2 reveals that while an increment in payload size has a marginal impact on the mean response delay, it considerably escalates the 95th percentile. This observation underscores an amplification in jitter attributable to the enhanced payload size.

Furthermore, diminished request intervals are correlated with a pronounced elevation in the mean application response delay. This is attributed to the augmented data rate on the uplink caused by reduced request intervals, a phenomenon that also precipitates a significant surge in the 95th percentile of the recorded measurements.

Conclusively, it is evident that although both reduced request intervals and enlarged payload sizes contribute to an elevated uplink data rate, the former exerts a more pronounced influence on the mean response delay of an HTTP application operating on the edge of a 5G network. This analytical insight is integral for optimizing the balance between data throughput and response time, pivotal for enhancing the user experience and operational efficiency of applications run over 5G networks.

7.2.2 Application performance with DRX enabled

In Figure 7.3, we evaluate the variance in response delay of HTTP applications attributable to DRX amidst escalating request intervals. Our observations from Figure 7.3 do not indicate a pronounced effect on both the average response delay and its 95th-percentile for the appraisals made on the HTTP application. Similar to scenarios where DRX is not operational, a marked decline in the average response delay is noticeable when the request interval is below 20ms; this trend, however, stabilizes as the interval expands beyond this point.

The muted impact of DRX can be primarily ascribed to the inherently dynamic nature of most control plants. These entities frequently exhibit high request frequencies that are in stark contrast to the more protracted DRX cycles in 5G systems. The extent of

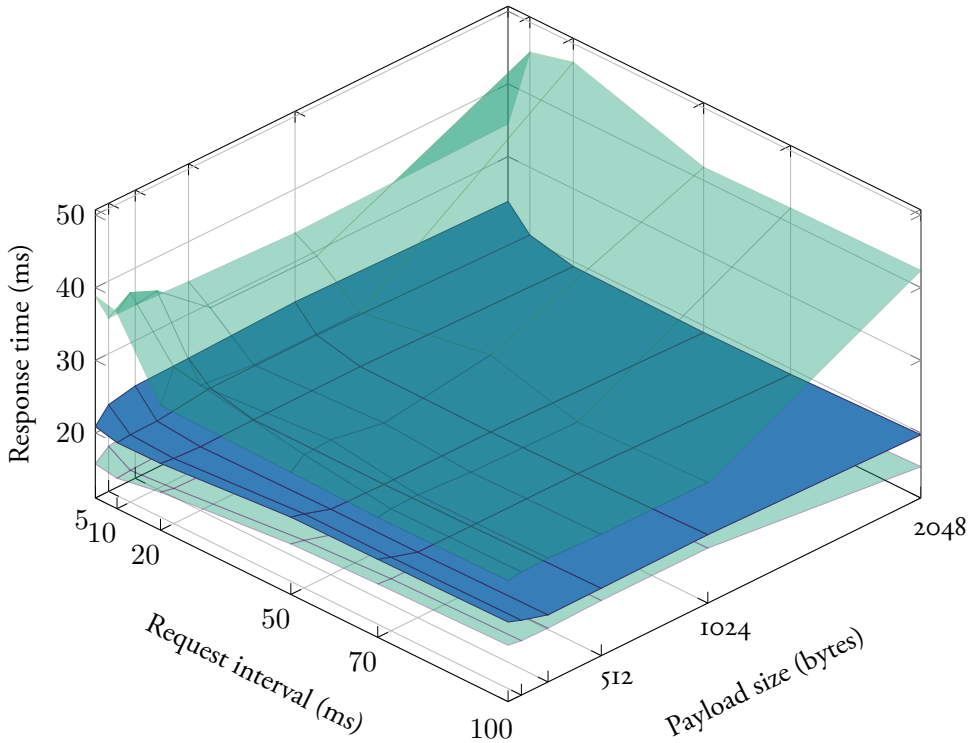


Figure 7.2: Impact on HTTP response delay of request intervals and payload size

delays induced by the DRX mode depend on a blend of the request frequency and the length of the DRX cycle. In our analysis, we observed that the HTTP application–serving as a representation of various time-critical control processes–ensures that the UE remains active and connected due to the high frequency of its requests. This constant connectivity effectively reduces the potential issues that could be caused by the DRX mode.

7.2.3 BnB process when running

In Figure 7.4, we present the step response of the outlined BnB process alongside the response delay experienced when the controller is (1) connected directly to the plant via a wired network and (2) situated at the edge of the 5G network. The upper segment of Figure 7.4 depicts the dynamic position of the ball's position as the set point oscillates between the two ends of the beam. The corresponding lower segment provides a visual representation of the response delay associated with each request

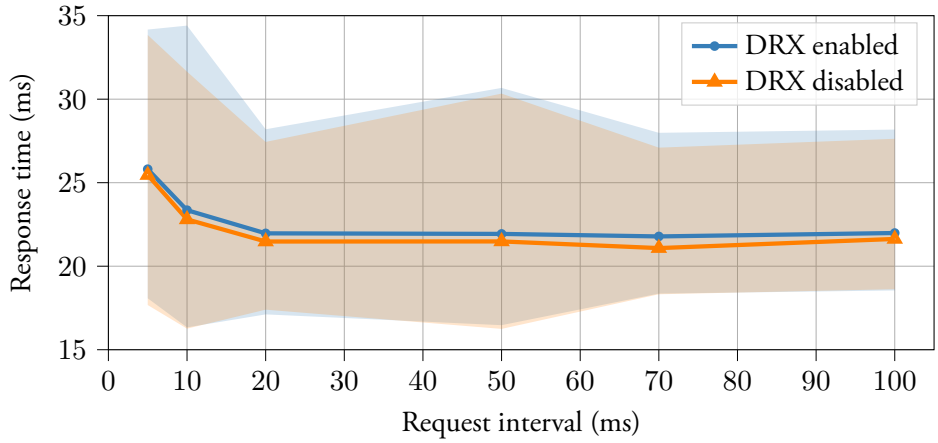


Figure 7.3: Mean values of response delays when enabling and disabling DRX

dispatched from the BnB plant under both operational scenarios.

A close examination reveals that the intertwined operations of the controller's computations and the plant's processing contribute to a roughly 20ms delay in state updates and control signal computations. The integration of the 5G network and edge computing introduces an additional delay, approximately 30ms, accompanied by a noticeable increase in delay variance.

However, as illustrated in the step response detailing the ball's positional adjustments, the BnB plant adeptly navigates to the designated set points even with the added delays attributed to the 5G and edge integration. This underscores the system's resilience and effectiveness in maintaining control integrity amidst the extended response delays to receive control signals.

7.3 Conclusion on the 5G-assisted CCS

This chapter provides an insightful exploration into the deployment of Cloud Control Systems aided by a mid-band SA 5G RBS. We rigorously evaluate the end-to-end response delay to gauge the viability of deploying a CCS over the 5G network and its affiliated edge environment. The deployed CCS process yielded stable outcomes, courtesy of a robust 5G network system and a finely-tuned controller.

However, our extensive evaluation reveals that integrating mobile communication systems and edge computational processes contributes significant additional delays to the application's response time. While these delays didn't undermine the performance

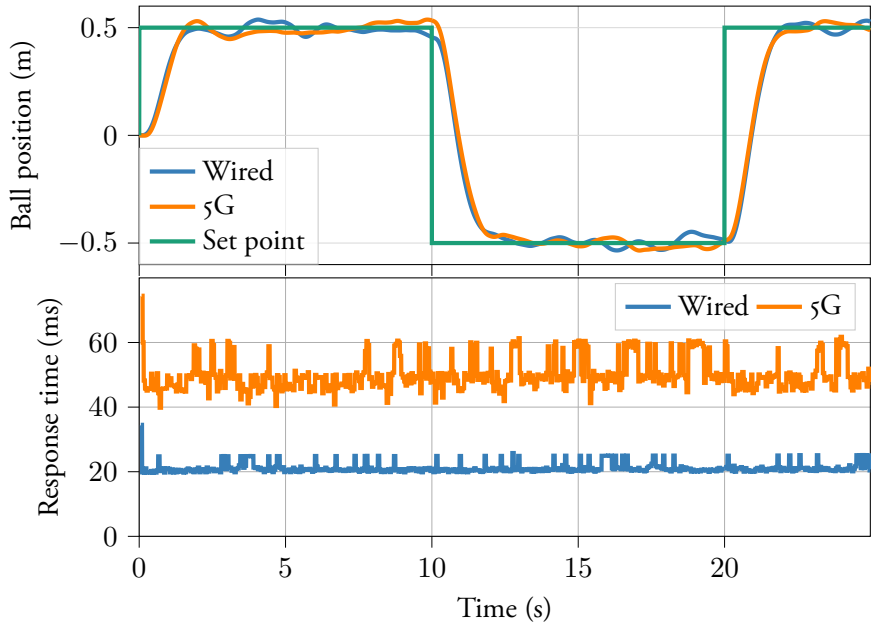


Figure 7.4: Step response of BnB process with different controllers

of a BnB plant operated under MPC, they could potentially jeopardize the stability of more time-sensitive control plants or those managed by less optimized controllers.

These findings accentuate the need for in-depth research aimed at the refinement of CCS implementation strategies, encapsulating an array of factors intrinsic to network and computing infrastructures.

As we advance to the next chapter, our focal point transitions to the comparative analysis of distinct transport layer protocols and their associated application protocols in the context of CCS deployment. The core of this comparative study is anchored on QUIC versus TCP, with the primary objective being the identification of a more efficacious protocol that stands resilient under a diverse spectrum of network and computational settings.

Chapter 8

Transport layer protocols in Cloud Control Systems

This chapter elevates the discourse to the transport layer, on a comparative analysis of two predominant transport layer protocols that are integral in the contemporary landscape of cloud and web applications: QUIC and TCP.

8.1 Motivation

Our motivation to evaluate transport layer protocols within the context of CCSs originates from the observed limitations associated with TCP and HTTP1.1 in previously conducted studies. Notable challenges such as HOL blocking and protracted handshakes prompted an exploration for an alternative protocol that could overcome these challenges and still fit well with the fast-paced, efficient world of cloud computing.

Enter QUIC—a protocol birthed and refined by Google, seen as a potential successor to TCP in the realm of web application deployment. It has been proven to provide lower latency and fewer request errors in numerous applications under the aegis of Google and Meta. [Goo; JC].

In the context of CCSs, QUIC also emerges as a promising candidate, potentially offering advantages of reduced latency and enhanced network performance—attributes that are pivotal in optimizing the QoS of a CCS.

Consequently, this chapter is orchestrated to unravel the comparative efficacies of QUIC and TCP, with an overarching goal to discern the protocol that not only aligns

with but amplifies the QoS of a CCS. Our investigation takes a close look at the complex aspect of each protocol, in order to offer a comprehensive perspective that informs decision-making in the deployment of CCSs in a cloud-native environment.

Before delving into the evaluation, it is essential to clarify specific terms and definitions pertinent to this chapter to ensure a comprehensive understanding. The following are explicit definitions:

- **QUIC**: This refers explicitly to the IETF-standardized QUIC, which aligns with the principal QUIC implementations available in various programming languages.
- **HTTP1**: This term denotes HTTP1.1, accompanied by TLS1.2, as applied within this chapter's context.
- **Connection**: This pertains to the transport layer connection. More specifically, it encompasses TCP, UDP, and QUIC connections. It is noteworthy that a QUIC connection operates over a UDP connection.
- **Stream**: A stream is characterized as a unidirectional or bidirectional channel, transmitting ordered data bytes within a transport layer connection.
- **Payload**: This refers to the application data or the highest layer of a packet's payload. In the absence of an implemented application layer (HTTP), it represents the payload of the TCP or QUIC packet. Conversely, with HTTP in place, it is the payload of the HTTP packet.

8.2 Experiment Setup

In this section, we delineate the experimental framework and metrics employed to examine the performance of distinct protocols within CCSs. Our analysis primarily focuses on QUIC and TCP, exploring their performance as transport protocols. With HTTP's ubiquitous presence in cloud applications, our evaluation extends to assessing the performance dynamics of HTTP3 and HTTP1.1; the latter is layered over TLS and TCP.

Accordingly, we have orchestrated an examination consisting of four distinct cloud applications, each operating under a unique protocol as outlined below. This methodology is designed to provide a comprehensive and detailed analysis, illuminating the complex performance attributes vital for making informed choices regarding protocol selection in the realm of CCSs.

1. QUIC (single stream)
2. TCP (without TLS)
3. HTT3 (HTTP over QUIC)
4. HTTP1 (over TLS/TCP)

It's essential to note the distinctions in the protocol configurations. In the first two cases, we are not implementing any application protocols. Consequently, the packet size and network overhead are reduced compared to cases 3 and 4. Moreover, the absence of multiplexing in the single-stream QUIC and TCP configurations results in head-of-line blocking under extensive network latency.

For case 3 and 4, we implement HTTP persistent connection and asynchronous requests. QUIC inherently handles this, allowing the initiation of new streams within a single UDP connection for each client request, facilitating multiplexing. In contrast, HTTP1.1 necessitates application-level management of asynchrony. When a subsequent request is initiated before the arrival of a response to a preceding request, a new TCP connection must be established to transmit the request. This approach ensures multiplexing and prevents blocking associated with the prior connection.

Our evaluation methodology mirrors the approach delineated in Section 7.1.1, although it is implemented in Golang. We establish a series of client-server pairs communicating via diverse protocols. In this setup, the client represents the plant of a control system, periodically transmitting its current state to the server, which embodies the remote cloud controller. The QUIC and HTTP3 protocols are implemented utilizing the `quic-go` library[1], adhering to the IETF standard for the QUIC protocol.

The client is executed on a machine running Ubuntu-jammy, equipped 5.19.0-38-generic Linux kernel. In contrast, the servers are encapsulated within Docker containers and are allocated across two distinct environments: an edge Kubernetes cluster and a public cloud infrastructure. We also perform the analysis under three distinct networking scenarios, determined by the servers' deployment locales. These scenarios encompass:

1. Ethernet (wired LAN) - A direct, wired connection ensuring optimal speed and minimal interference.
2. Private 5G mobile network - A sophisticated, high-speed mobile network enabling real-time data exchange.

[1] `https://github.com/quic-go/quic-go`

Table 8.1: Three evaluation scenarios in the experiment

Scenario Name	Ethernet	5G	AWS
Deployment Env.	Edge cluster	Edge cluster	AWS Public Cloud
Network	Ethernet LAN	5G	Internet
RTT	0.520ms	10.688ms	13.553ms

3. Internet - A universal yet complex environment introducing variable factors such as latency and potential data loss.

Our experiments are thoughtfully constructed to evaluate protocol efficiency in untouched, realistic network environments. Consequently, we avoid making any specialized adjustments or tunings to the networks, aiming to preserve and examine their natural, unmodified conditions.

The edge cloud setting for our experiments is provided by the lab Kubernetes cluster, detailed in Section 2.1.4. Every server operates within a Pod characterized by BestEffort QoS class. Under the edge scenario, two distinct networks are employed. The first is a straightforward, wired network ensuring direct Ethernet connections between the client and the edge cluster. The second incorporates the 5G mobile network, thanks to the LTH 5G RBS, as elaborated in Section 2.2.3. The client machine engages this network via a 5G WNC modem.

In the public cloud scenario, each server is manifested as an Elastic Container Service (ECS) on AWS, hosted in a data centre in Stockholm, Sweden. Every service is allocated 1 vCPU and 3GB memory, facilitated by AWS Fargate. In this scenario, communication unfolds over the Internet, introducing a level of unpredictability and complexity not seen in edge scenarios, as data exchanges are subject to the inherent variances and challenges of online interactions.

Table 8.1 summarizes the specifications and distinctions of each scenario. To enrich our analysis and provide a more comprehensive understanding of the network dynamics, we've included RTT values in the table. These were determined using Internet Control Message Protocol (ICMP) "PING" measurements, offering a detailed view of the latency characteristics inherent to each respective network.

Similar to the previous chapter, we mimic the behaviour of a CCS by designing a communication pattern between each pair of client and server in our experiment. Specifically, our client sends a request to the server at intervals of $d$ms, emulating the sampling rate of a control system. Each request carries a payload of p bytes, and the server responds with an equivalent payload. This setup simulates a control system operating with a $d$ms sampling time and p byte message exchanges between the plant and the controller. Our tests employed various d and p values to replicate

the conditions of different types of control systems. We ran each set of parameters for an hour to gather a robust dataset for our analysis.

- d (ms): 5, 10, 25, 50, 75, 100
- p (bytes): 128, 256, 512, 1024

8.2.1 Performance Metrics

Rather that performance metrics directly related to control processes, we focus on the performance of the communications in a CCS, which has pivotal influence on the QoS performance of a CCS and the cost of the system deployment. Hence, three metrics are discussed in our protocol evaluation:

- response delay
- jitter of response delay
- data volume.

The **response delay** is measured at the plant, denoting the time elapsed between initiating a request and receiving a corresponding response from the server. This metric is crucial as it underscores the plant's efficiency in receiving and responding to control signals in real-time. It's pivotal to note that the response delay often surpasses the network latency, encapsulating the processing times attributed to the transport, application protocols, and the execution duration of the controller application within the cloud.

We quantify the **jitter** of the response delay as a percentage, derived by dividing the mean jitter value by the mean response delay, as outlined in [Por+06]. Jitter becomes a critical consideration in a CCS especially in instances where the network latency aligns with or exceeds the sampling rate of the control process. Such scenarios often call for advanced compensatory measures like state predictions to compensate delays. Elevated jitter levels indicate a degree of unpredictability in network and plant operations, complicating the implementation of delay compensatory measures. Conversely, lower jitter levels indicate a level of predictability, facilitating more straightforward delay compensations.

Data volume assessment becomes indispensable, particularly in setups where the control is executed within a public cloud and the incurred service costs are contingent on the traffic generated. We choose evaluation criteria centred on the average packet size

for each request and response (Equation (8.1)), as opposed to throughput which is inherently tied to network bandwidth and data rate–-factors influenced by the control plant frequency and payload size. Our emphasis on average packet size is intended to reveal the network overhead linked to each protocol, enabling decisions that balance reduced data volume and maintained performance quality.

$$\begin{aligned} \text{data volume per request} &= \frac{\text{Total sent data by the plant}}{\text{Number of requests}} \\ \text{data volume per response} &= \frac{\text{Total received data by the plant}}{\text{Number of responses}} \end{aligned} \tag{8.1}$$

8.3 Evaluation Results

In this section, we detail the outcomes of our evaluation and provide an analysis of the results.

8.3.1 Response Time and Jitters

The evaluation of response delay and its jitter is categorized into two distinct scenarios based on the designated values of the parameter d and the network latency, as outlined in Table 8.1:

- Low-frequency Case
- High-frequency Case

In the low-frequency case, the value of d exceeds the network latency. An example of this is when $d = 25$ms in AWS and 5G scenarios. Under these conditions, while the average response delay is notably extended beyond network latency, it typically remains beneath the span of one sampling interval d. In such a configuration, the performance metrics of response delay and jitter predominantly reflect the network's reliability and consistency within the system's operational framework.

In the high-frequency case, the sampling interval d is reduced to a span that is less than the network latency. This contraction in interval amplifies the need for delay compensation in the control system. The key objectives become reducing response delay and minimizing jitter, essential steps to optimize both system and controller design. However, the challenge grows as one must carefully navigate the constraints associated with the increased frequency.

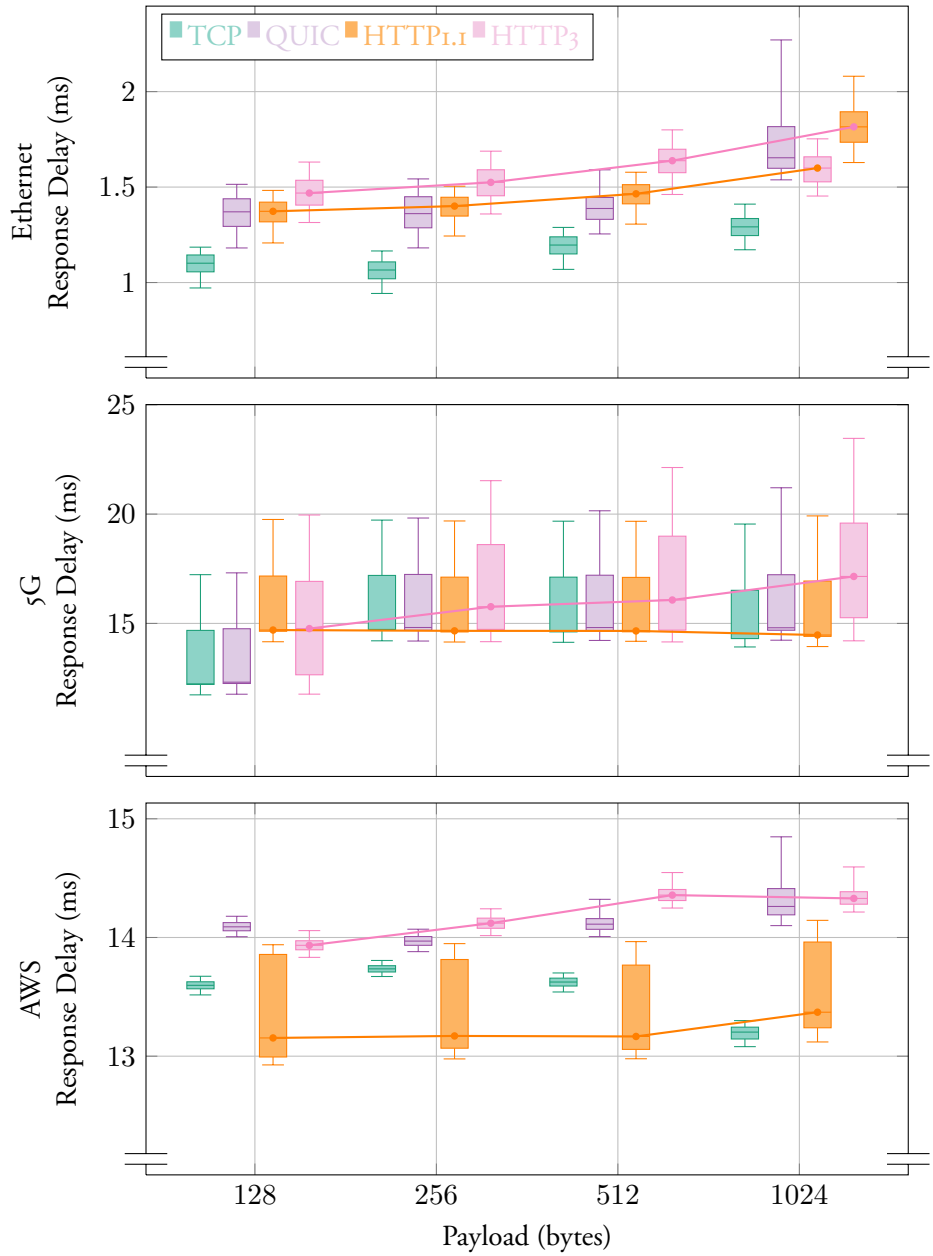


Figure 8.1: Box-plots illustrating the response delays of various protocols under Ethernet, 5G, and AWS scenarios, considering an increasing payload size (p) and a consistent sampling rate ($d = 25$ms). Given the varied network latency scales inherent to each scenario, the plots are rendered with adapted scales. The y-axes, deviating from their origin, accommodate these variations to provide a clearer visual representation.

Low-frequency case

Figure 8.1 presents box plots illustrating the response delays associated with different protocols across Ethernet, 5G, and AWS scenarios. Each plot delineates the 5th, 25th, 50th, 75th, and 95th percentiles of the collected data, from bottom to top. The analyses were conducted with an incrementing payload size and a constant sampling rate ($d = 25$ms); results for cases with sampling rates $d \geq 25$ms are similar to these findings.

In all scenarios, single stream QUIC exhibits approximately 0.3ms higher delays than TCP. This discrepancy can be attributed to the absence of cryptographic implementation in our TCP setup, which reduces both handshake latency and processing time relative to QUIC. The trend of increasing response delays with enlarging payload size is observable, as a consequence of the augmented processing demands of larger packets.

In the Ethernet scenario, which is characterized by its reliable, lossless network, HTTP3 underperformed relative to HTTP1.1, echoing the conclusions of preceding studies [Seu+19; SLM20]. This pattern persisted in the 5G tests, affirming the network's reliability and minimal loss profile. A noticeable sensitivity of QUIC and HTTP3 to payload size was evident, particularly impacting the median response delay.

In the AWS scenario, the behaviour of protocols notably deviates from that observed in the other environment, with HTTP1.1 displaying almost a 1ms reduction in median response delays compared to TCP and exhibiting a broader interquartile range. This anomaly is potentially attributed to the cloud providers' traffic management policies or security protocols that influence specific types of traffic, specifically pure TCP and QUIC without an accompanying application protocol, being restricted or deprioritized. Additionally, a unique aspect of the AWS environment is the differential treatment of packet decryption. HTTP1.1 packets with TLS/TCP remain encrypted, while those transmitted via HTTP3 and QUIC are decrypted while our data collection, likely due to stringent security protocols in the data centre, influencing the delay dynamics associated with HTTP1.1 in this specific context.

AWS also exhibited diminished response delay variances compared to Ethernet and 5G, a phenomenon further elaborated in Figure 8.2. This consistency, though beneficial, is partly due to the relatively long network latency under this scenario, as well as the opaque nature of the execution and networking environment within AWS's ECS, where user control is limited, and certain operational details remain unknown.

In the low frequency case ($d \geq 25$ms), HTTP1.1 and HTTP3 operate over a single, persistent connection at the transport layer, TCP, or UDP, respectively. HTTP1.1

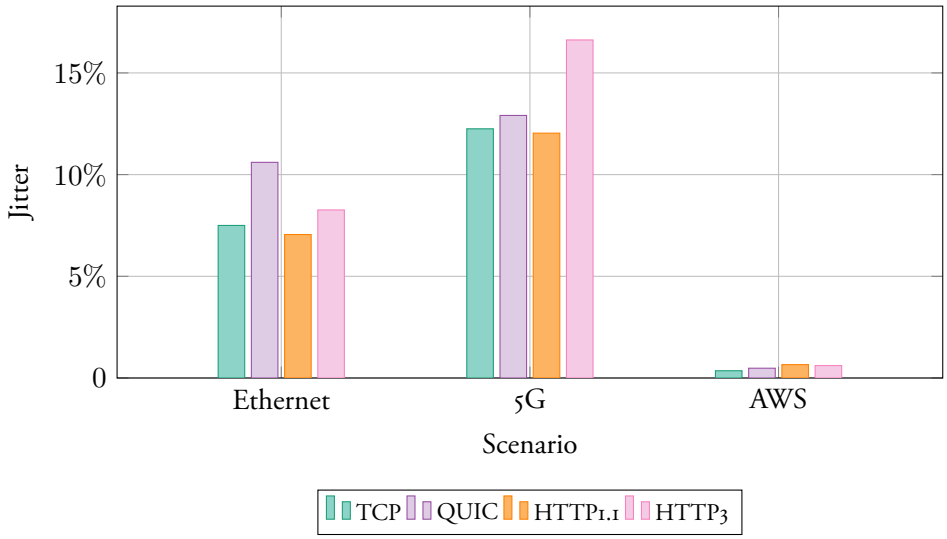


Figure 8.2: The jitter of different protocols under three scenarios, with payload size $p = 256$bytes and sampling rate $d = 25$ms.

benefits from the persistent and reusable nature of TCP connections, reducing the overhead of repeated handshakes. In contrast, HTTP3, while operating over a single UDP connection, initiates a new bidirectional stream for each request-response pair. This mechanism, while efficient, introduces an overhead as the server must consistently send MAX_STREAMS QUIC frames to allow the cumulative opening of new streams. This process, paired with the separate acknowledgments for each stream, enlarges the response delays and augments jitter, marking a distinct operational characteristic of HTTP3 in low-frequency communication systems.

High-frequency cases

In scenarios characterized by a high-frequency case, where network latency surpasses the plant sampling interval d, protocols necessitate a multiplexing function for optimal performance. The comparison here is streamlined to HTTP3 and HTTP1.1, as pure TCP or single stream QUIC are unequipped for such tasks. A representative illustration of this is presented in Figure 8.4, showcasing the performance of the two protocols at a 5ms plant sampling interval amidst an average network latency of 10.688ms, as observed in the 5G scenario. The choice of the 5G environment for this analysis is anchored on its higher latency profile compared to Ethernet and a more user-manageable computational setting than AWS.

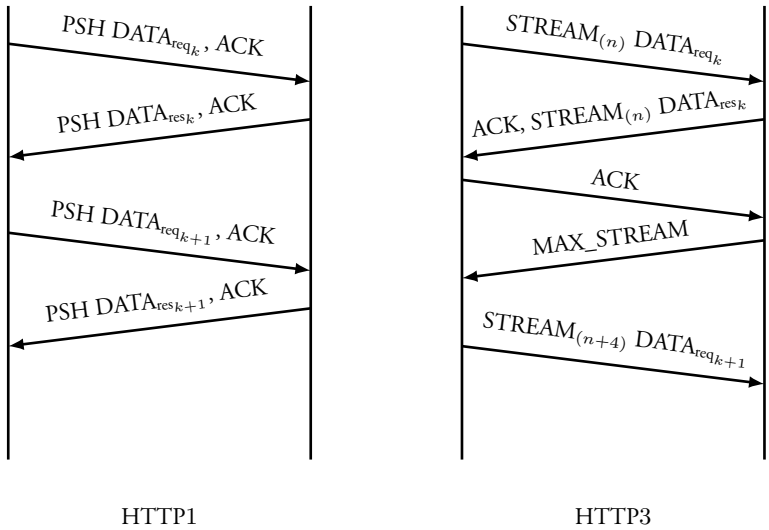


Figure 8.3: Data packet flows in a single HTTP1.1 persistence connection and HTTP3.

A noticeable shift in performance dynamics is evident here, with HTTP3 displaying enhanced efficiency in terms of both response delay and jitter, a departure from its performance in the low-frequency context. In situations where the network latency exceeds the plant's sampling interval d, HTTP1.1 encounters challenges due to head-of-line blocking. This issue manifests when delayed packet inhibit the entire connection, culminating in escalated response delays and jitter. In order to mitigate this, applications leveraging HTTP1.1 can initiate a new TCP connection for ensuing requests, even before the arrival of responses to preceding ones.

HTTP1.1's strategy to counter HOL blocking, however, is not seamless. It involves opening new TCP connections for upcoming requests if the responses for previous ones are yet to be received. This process, while functional, can lead to an increased establishment of sessions and potentially, an elevated overhead due to repeated transport and cryptographic handshakes. This manner is depicted in Figure 8.5, where the number of established sessions per 1000 requests rises in conjunction with increased communication frequencies, intricately influenced by the dynamics of network latency and jitter. HTTP3, conversely, exhibits an inherent advantage by permitting multiple streams concurrently for several requests without the necessity for additional handshakes, echoing its operation in low-frequency scenarios. This capability underscores HTTP3's superior performance metrics in high-frequency environments, particularly in response delay and jitter.

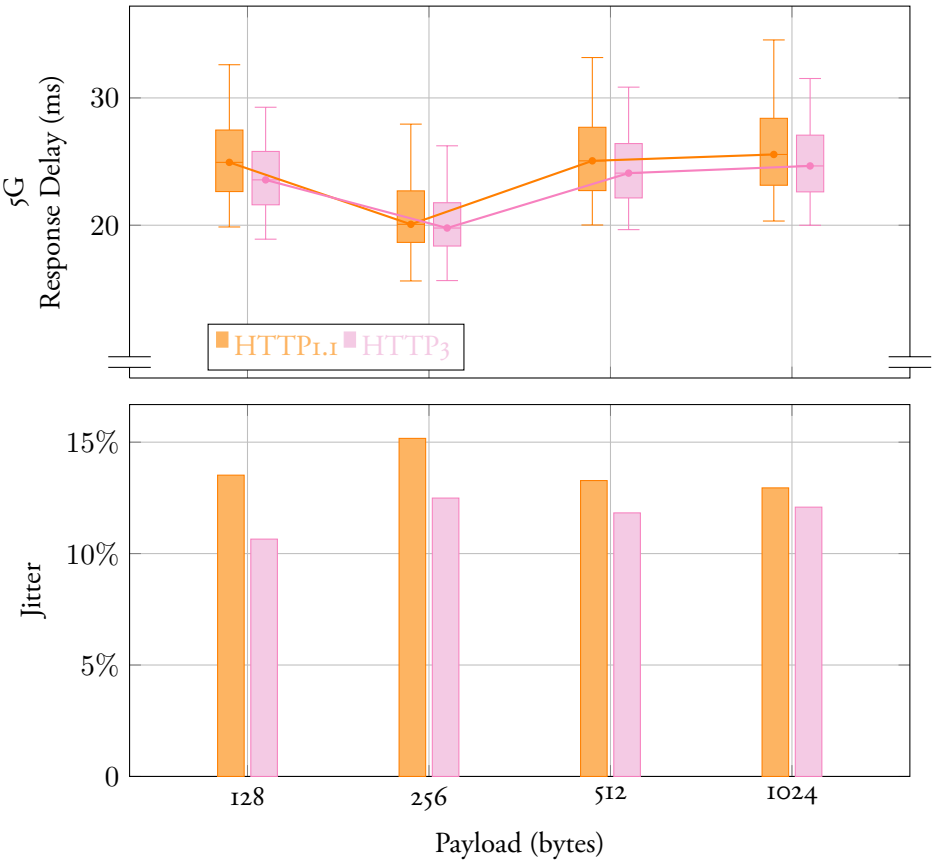


Figure 8.4: Comparison of response delay (top) and jitter (bottom) performance between HTTP1.1 and HTTP3 under the 5G scenario with increasing payload size and a sampling rate of $d = 5$ms. The y-axis of the response delay performance is broken from origin start.

8.3.2 Data Volume

In Figure 8.6, we display the data volume outcomes from experiments conducted under the 5G scenario, with each request and response carrying a payload of 128 bytes. This scenario is selected for its higher network latency and the controllable nature of the edge cloud environment. The results reveal that HTTP1.1 incurs a more substantial data volume per request compared to HTTP3 across all test cases. This increase is attributed to TCP's larger header size compared to UDP and QUIC, leading to an enlarged packet size for HTTP1.1 even when the payload remains constant.

A comparison of the data volume per request and the actual HTTP packet sizes underscores that HTTP1.1's data volume aligns more closely with its actual packet size

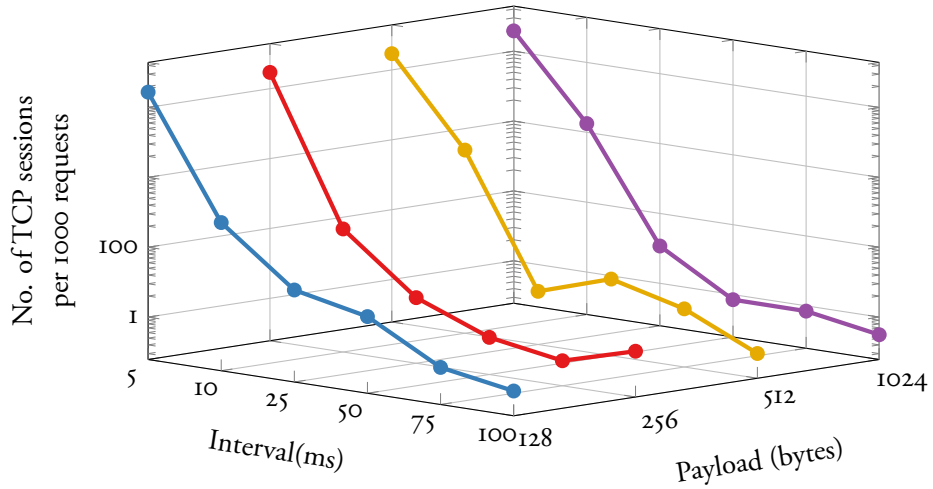


Figure 8.5: Average number of TCP sessions established in HTTP1.1 experiments for 1000 requests under different plant sampling intervals and payload sizes. The z-axis value indicates the number of sessions per 1000 requests. In the higher frequency case with a sampling interval of $d = 5$ms, approximately 100-200 sessions were established for every 1000 requests. This number decreased to less than 15 sessions per 1000 requests when the sampling interval was $d = 10$ms. When d was greater than 50ms, typically only 1 or 2 sessions were used during each experiment.

than HTTP3's does. This suggests that HTTP1.1's overhead is predominantly a result of the protocol's substantial header size, whereas HTTP3's overhead stems from stream control frames and ACK packets, as previously mentioned.

A noteworthy observation is the significant spike in HTTP1.1's data volume in the high-frequency case, where the sampling rate is $d = 5$ms. This surge is a consequence of the frequent TCP session establishments characteristic of this scenario, as evidenced in Figure 8.5, leading to an inflated network overhead from handshake packets. Generally, when considering the costs associated with public cloud controller deployment, HTTP3 emerges as a cost-effective alternative to HTTP1.1, owing to its reduced data volume footprint.

8.4 Conclusion on CCS performances over different protocols

This chapter delves into the exploration of the potential advantages that QUIC/HTTP3 holds over traditional TCP-based protocols within the context of CCSs. Our experimental design, covering three distinct network scenarios, centres around the assessment of protocols based on pivotal parameters: system response delay and jitter, and the data volume incurred.

A notable finding is the economic efficiency of HTTP3, underscored by its reduced

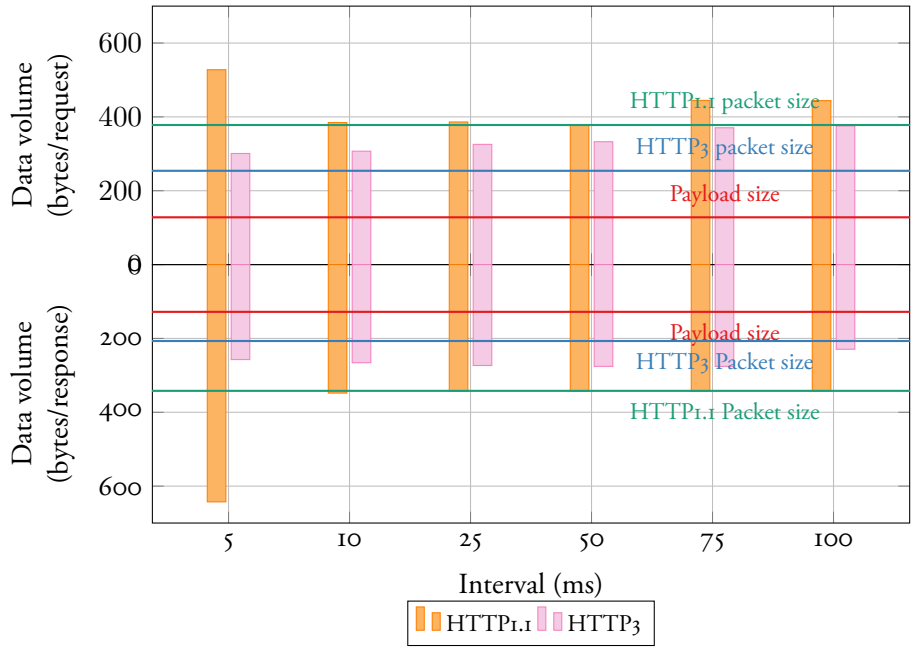


Figure 8.6: Data volume per request (top) and per response (bottom) under the 5G scenario with different sampling intervals and payload size of 128 bytes. The bar charts depict the data volume generated by different protocols, where the red lines represent the payload size of 128 bytes. Additionally, the orange and blue lines denote the data volume generated by a single HTTP1.1 and HTTP3 request and response.

per-request data generation. This attributes to its potential as the preferred choice for public cloud controller deployments, marking a shift towards cost-effectiveness.

When exploring the latency landscape, an interesting dynamic unfolds. In scenarios characterized by low-frequency control systems and negligible network latency, QUIC and HTTP3 lag behind TCP and HTTP1.1 in performance–a consistency echoed in some existing literatures. However, the narrative shifts in high-frequency contexts where network latency is significant; here, HTTP3 excels, showcasing reduced and more consistent response delays. This characteristic positions it as a viable candidate for time-sensitive applications and intricate control designs.

The granular analysis of protocol performance in high-frequency scenarios reveals a nuanced advantage of HTTP3 in CCSs, underscored by its reduced latency and jitter —even though the margin of improvement over HTTP1 is marginal, clocking at just 2 or 3 milliseconds.

As control systems are obligated to regularly send updates on the plant's state, the role of network latency becomes paramount, especially when it exceeds the plant's

sampling interval d. In this context, the plant is compelled to dispatch the next batch of state data, irrespective of whether a response to the preceding request has been received. This pattern leads to a scenario where control signals are expected to be acted upon before the emission of subsequent state updates, a situation that is complicated further when the arrival of expected control signals is delayed.

To mitigate these challenges, the implementation of intricate delay compensation strategies becomes essential, a topic that we will delve deeper into in Part IV of this thesis. In this domain, the virtues of HTTP3 become prominently beneficial. The protocol's diminished latency and jitter are instrumental in refining the precision of delay estimations, a factor that is central to the optimization of sophisticated control systems.

Thus, from the perspective of enhancing control performance in environments where time sensitivity is a critical attribute of CCSs, the conversation favours HTTP3. The protocol emerges as a potentially revolutionary tool capable of elevating operational efficiency and the granularity of control, heralding a new era of enhanced performance and reliability in Cloud Control Systems.

Summary

This part outlines our implementation of Cloud Control Systems (CCSs) and the ensuing evaluation of system performance under various protocols, network conditions, and computing environments. Our results highlight a promising performance that aligns with the network QoS requirements of diverse CCSs. However, it is important to note that the majority of these tests were conducted using a basic client-server model, without a deep dive into the nuanced performance aspects of an actual control system.

Indeed, there are instances where the control task may encounter failures due to system constraints. A representative example of this is the deployment of a time-sensitive application operating at a high frequency across a network characterized by significant latency. In such scenarios, the limitations of the current setup become evident, necessitating enhanced solutions to ensure seamless operation.

As we progress to the subsequent part, our focus will shift towards devising strategies to augment the robustness and efficiency of CCS operating in the unpredictable terrains of wild cloud and networks. We aim to enhance the system's adaptability, ensuring that it delivers optimal performance even when deployed over networks with inherent variability and unpredictability. This initiative is particularly pertinent for applications of Cloud RAN and CCS that are dependent on the reliability and responsiveness of network infrastructures to meet their operational mandates.

In essence, our forthcoming exploration is dedicated to uncovering and implementing remedies that will elevate the resilience and performance of cloud integrated systems operating in diverse and often challenging network and cloud environments, ensuring that they not only meet but exceed the anticipated QoS benchmarks.

Part IV

Punctual Cloud for Time-critical Cloud Integrated Systems

Introduction

In the preceding part, we engaged in a detailed exploration of systems controlled via wild networks and the cloud. This part advances that narrative, focusing acutely on a pivotal challenge: system-induced delay. We introduce the concept of **punctuality** as a foundational element to ensure optimal functionality in real-time and time-critical systems operating in cloud environments.

Real-time applications are anchored in the promptness of responses from their computational tasks. Delays, especially those protracted, can result in outdated actuation signals leading to potential system failures. Conventionally, such applications are housed close to the process's consumer, ensuring immediacy in responses. However, the cloud environment, marked by its network separation, unpredictable virtualization, and complex software layers, introduces inherent delays and disturbances [GCL14; Ha+16; WN10; Dre08; Lar+20].

While the cloud, especially its edge and fog architectural variants, has shown promise in accommodating real-time applications, yet the inherent and variable delays pose challenges in mirroring the responsiveness of on-board computing environments, as detailed in Chapter 3. Given the financial and technical impracticality involved in transforming existing infrastructures for ultra-low latency communication, we propose an adaptive approach for applications, positioning them to naturally accommodate the cloud's latency landscape.

In this progression, we unveil the **Punctual Cloud** framework, a novel approach engineered to guarantee timely response deliveries amidst the inherent cloud-induced delays. This part is tailored to provide nuanced insights into the application of this framework in both Cloud RAN and Cloud Control Systems (CCSs), accepting the unavoidable and stochastic nature of network latency and crafting strategies to minimize its impacts.

Our discussion commences with an in-depth analysis of the punctual cloud framework's integration within a Cloud RAN system, with a special emphasis on resource

allocation challenges. We begin in the confines of a simulated environment to grasp preliminary insights and understandings. This foundational knowledge from the simulation paves the way for the framework's application in a real-world context, utilizing a Kubernetes cluster in our lab to bring theory into practice.

Following this, the focus shifts to the implementation of the punctual cloud framework in a CCS. Here, the BnB control plant, previously explored in Chapter 7, serves as a practical example to illustrate the framework's adaptability and efficacy in real-time operational settings.

It's important to remember that there is not a "one size fits all" approach when it comes to applying our framework. Each deployment is a bit different, tailored to fit the unique needs and characteristics of every system, whether it is being tested in a simulation or applied in a real cloud environment. These customizations are necessary because each system has its own specific needs and QoS requirements.

But even with these differences, there is a common element that ties all deployments of our framework together - its core logic. It is like having a universal remote that can be easily tweaked to control a variety of devices. This means that, with just a few small adjustments, the framework can be applied to a broad range of cloud integrated systems, ensuring they respond in real-time and perform at their best, no matter the environment they are in.

Chapter 9

Punctual Cloud for Latency-aware Resource Allocation in Cloud RAN

Building on the discussions in Chapter 6, this chapter revisits the resource allocation problem of Cloud RAN, specifically within the context of massive MIMO uplink pilot scheduling in an Industry 4.0 environment. The difference of this system from Chapter 6 is that, it addresses the stochastic nature of a Cloud RAN system in the resource allocation problem, and implements the system in cloud infrastructure in a microservice paradigm.

In this chapter, we roll out the punctual cloud framework as a specialized radio resource allocation strategy for Cloud RAN. This strategy is designed to mitigate the impact of the unpredictable delays inherent in cloud environments. Alongside this, we also introduce a simulation model of the Cloud RAN system, using it to evaluate the effectiveness of our framework in an Industry 4.0 context.

Our discussion continues with a real-world implementation of this framework within our lab's Kubernetes edge cluster. Our empirical findings indicate that adopting the resource allocation mechanisms of the punctual cloud framework substantially improves the system's radio resource utilization. Importantly, this optimization is achieved without sacrificing the reliability of communications. Thus, the punctual cloud framework offers a robust and flexible solution to the challenges posed by cloud-induced variability.

9.1 Targeted System

The operational context for this investigation mirrors that of the Cloud RAN system outlined in Chapter 6 and depicted in Figure 6.1. We focus particularly on an indoor factory automation setup where UEs, also referred to as industrial units, interact via a network managed by Cloud RAN. Given that these industrial units predominantly comprise industrial IoT devices, their density can be substantial—up to 10000 devices per km^2 according to [Gro19].

In the Cloud RAN system, a scheduler operates in real-time to distribute available radio resources to these industrial units, guided by an allocation policy. The primary aim of such policies is generally to mitigate resource starvation, collision, and network congestion. In this Cloud RAN system, allocation decision-making, also called as the scheduler, is performed at the BBU pool and subsequently executed by the RRH.

As we delineated in Chapter 6, the two main types of industrial units are taken into accounts in our system: Critical Units (CUs) and non-Critical Units (non-CUs). While our focus remains on the CUs for the resource allocation challenge, non-CUs entities are treated as background traffic. The pilot scheduling predominantly prioritizes transmission requests from CUs, only addressing non-CUs once all CUs have been adequately serviced during a coherence interval of the massive MIMO time-frequency space.

The scheduling operation, located in the BBU pool, is executed as a general-purpose cloud service. Typically, such services are backed by an array of worker nodes that share virtualized resources. A load-balancer then distributes incoming requests among these nodes. This design inevitably leads to a stochastic and dynamic character to Cloud RANs, much like any cloud service. This variability, termed "cloud delay", accounts for not just network latency but also request admission and execution times in the cloud. In the context of Cloud RAN, this randomness can inject disruptive delays into the signal processing chain of a BBU.

When it comes to radio resource allocation, these delays can trigger inaccurate assignments to industrial units. We discussed in Chapter 6 how this affects the trade-offs between resource utilization and transmission reliability. Consequently, there is a growing need for purpose-built scheduling algorithms capable of mitigating the impact of these system-induced disturbances.

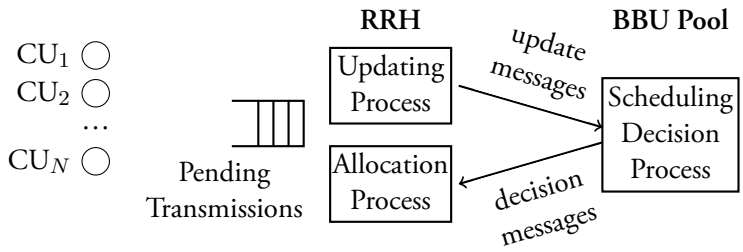


Figure 9.1: System model of targeted Cloud RAN system under punctual cloud framework.

9.2 System Model

This section details a model of the Cloud RAN system as structured under the punctual cloud framework, as illustrated in Figure 9.1. The core components of this system include a collection of CUs, as well as a Cloud RAN infrastructure that encompasses both a RRH and a BBU pool.

As indicated in Figure 9.1, **update messages** are transmitted from the RRH to the BBU within the Cloud RAN framework. The BBU, in turn, sends back a **scheduling decision**. Both these update and decision messages experience delays attributable to the stochastic nature of the cloud system.

9.2.1 Cloud Delay

Radio resource allocation over Cloud RAN includes information dissemination between the RRH and the BBU, as described in Section 9.2. Here we denote **update** message as the information sent by the updating process at RRH to the scheduling decision process resides in the BBU pool. Likewise, a **decision** message originates from the BBU pool to the allocation process at RRH.

The cloud, its opaque management systems, shared infrastructure, and intermediate network incur a stochastic delay. This delay is represented as two independent stochastic variables, T_{update} and T_{decision}, representing the time for making and delivering update and decision messages.

The two types of delays encompass all facets of time lags in the cloud environment, including execution times, admission and queuing delays. Additionally, these delays account for the time taken along the communication path of a given message. **Response delay**, as formulated in Equation (9.1), describes the time span between the sending of an update message and the receipt of the corresponding decision message.

$$T_r = T_{\text{update}} + T_{\text{decision}} \tag{9.1}$$

9.2.2 Industrial Units

In the model, we use N to represent the number of active CUs within the radio range of the RRH. Let CU_n stand for the nth active CU, where n is an integer belonging to the set $\{1, 2, ..., N\}$. Transmissions from each CU_n to the RRH are initiated according to a stochastic process, and the time interval between consecutive transmissions from CU_n is designated as c_n.

In this setup, a CU can only successfully transmit data during a coherence interval if it has been allocated a pilot signal. Each transmission initiated by CU_n carries with it a deadline, denoted as D_n. If a transmission is not allocated a pilot before reaching this deadline, it is considered a failed transmission and is subsequently discarded.

9.2.3 Massive MIMO Pilot Scheduling

During each coherence interval, the pilot scheduling process for the massive MIMO uplink allocates pilot signals to the available CUs. For broader context, we consider each coherence interval as a time **slot**, with its duration represented by T_s. We also assume that the RRH and BBU pool are synchronized in terms of time, meaning that a slot k signifies the same time interval for both entities.

As slot k commences, the RRH informs the BBU pool about its present state, specifically the number of pending transmissions for each CU_n, symbolized by $Q_n(k)$. We refer to $Q_n(k)$ as the state of CU_n. The collective state of all CUs at slot k is thus expressed as $\mathbf{Q}(k) = \{Q_1(k), Q_2(k)..., Q_N(k)\}$.

Upon receiving this information, the BBU pool carries out the pilot scheduling and sends the decision message back to the RRH, which then acts upon it. A scheduling decision intended for slot k is denoted by $\mathbf{P}(k) = \{P_1(k), P_2(k), ...P_N(k)\}$, defined as:

$$P_n(k) = \begin{cases} 1 & \text{allocate pilot to CU}_n\text{at slot } k \\ 0 & \text{not allocate pilot to CU}_n\text{at slot } k \end{cases} \tag{9.2}$$

During each slot k, the RRH allots pilots to active CUs as per the decision $\mathbf{P}(k)$. Let p be the total number of pilots available for each slot. Therefore, a maximum of

p CUs can be allocated pilots in a single slot. If $P_n(k) = 1$, then N transmissions from CU_n can proceed during slot k, making k the actuation slot for $\mathbf{P}(k)$.

9.3 Problem Definition

This section detail the complexities introduced by the stochastic behaviour of a Cloud RAN system in the context of the pilot scheduling process.

Initially, we examine the limitations of a **naive pilot scheduling scheme**, as employed and analysed in Chapter 6. This scheduling scheme does not factor in the inherent delays within the Cloud RAN infrastructure. In this setup, the scheduler is activated each time it receives an update message at the BBU pool. Upon processing, a corresponding scheduling decision is sent back to the RRH. Ideally, the complete cycle of receiving an update and sending back a decision should be accomplished within one time slot. This is because the RRH's state could change by the next slot, triggering a new update and subsequent decision-making cycle.

However, failure to account for the unpredictable delays in the delivery of update messages and scheduling decisions can result in timing issues. Specifically, a decision $\mathbf{P}(k)$ formulated in response to an update $\mathbf{Q}(k)$ might not be executed at the intended slot k in a timely manner. As illustrated in Figure 9.2, such stochastic delays can cause decisions to arrive either too late, after the corresponding update messages have expired, or in an incorrect sequence. This can result in inaccurate pilot allocation and deteriorate performance metrics, as observed in Chapter 6.

Therefore, it becomes imperative to design resource allocation algorithms that are robust to cloud-induced delays. In other words, irrespective of the variability in cloud delays, the Cloud RAN system should ensure that the majority of scheduling decisions are both timely and relevant, aligning well with the transmission requests they are intended to serve.

In our analysis, we primarily focus on three key performance metrics: **punctuality**, **pilot utilization**, and **reliability**. These metrics serve to assess the effectiveness of any given scheduling scheme within a Cloud RAN system.

9.4 Performance Metrics

Below we define the three performance metrics, along with a series of challenges based on the three properties.

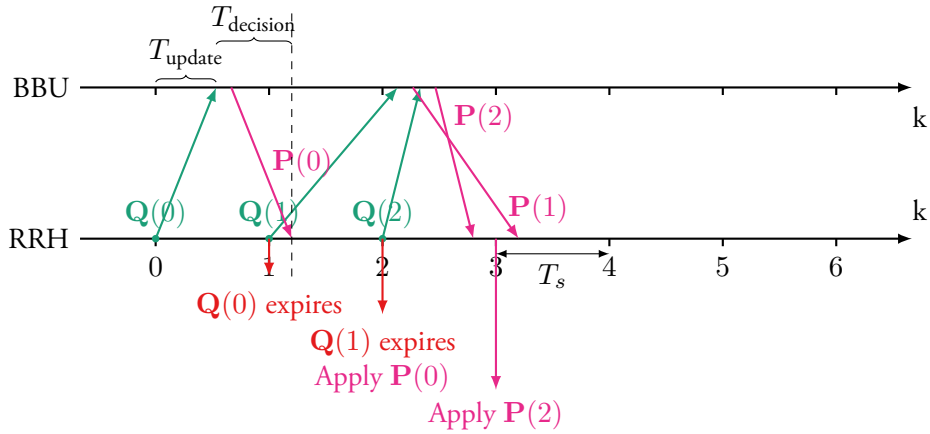


Figure 9.2: Consequences of stochastic cloud delays on subsequent update message $\mathbf{Q}(k)$ and underlying decisions $\mathbf{P}(k)$.

9.4.1 Punctuality (R)

As aforementioned, the "cloud delay" can make some decisions out-of-date by the time they are put into action. When this happens, we say the decision was "not timely applied". On the other hand, if a decision gets implemented right when it is supposed to, we call this a "timely applied decision".

Punctuality, denoted by R, is essentially how often decisions get applied on time. It's calculated as a ratio over a certain time frame. For example, between time slots k_i and k_j if we have got $\mathcal{K}$ decisions that were timely applied, then we can compute punctuality $R_{k_i:k_j}$ as Equation (9.3).

$$R_{k_i:k_j} = \frac{\mathcal{K}}{k_j - k_i} \tag{9.3}$$

With stochastic delays in the system, decisions can arrive late or even out of sequence, causing fewer of them to be "timely applied". When decisions are applied at the wrong time slots, the system can get confused about what resources are actually available, which can lead to mistakes. These errors can impact pilot utilization, reducing the overall efficiency of the system.

9.4.2 Pilot Utilization (β)

We use **pilot utilization** to evaluate the performance of the pilot scheduling strategy. Under the industrial scenario described previously, when the CUs and non-CUs co-exist and share the same mobile network, the non-CUs will get the remaining pilots in a slot when all transmissions from the CUs have been served. Therefore, unused, or wasted, pilots is highly undesirable. A pilot is **wasted** every time it is assigned to a CU that has nothing to transmit. This can occur if a scheduling decision is based on an outdated RRH state and if scheduling decisions are not delivered timely.

The decision $\mathbf{P}(k)$ determines which CUs will be allocated pilots at time slot k. Since there are only p pilots available in any given slot, the decision should adhere to the constraint $\sum_1^N P_n(k) \leq p$.

Assuming that a CU that is assigned a pilot can serve N transmissions, the actual number of pending transmissions for a specific CU_u at time is $Q_u(k)$. This leads to a certain number of wasted pilots for CU_u at that time slot, which we denote as $\omega_u(k)$.

$$\omega_u(k) = \max(0, (1 - \frac{Q_n(k)}{N})P_n(k)) \tag{9.4}$$

This leads to the pilot utilization $\beta(k)$ for all CUs in this allocation:

$$\beta(k) = 1 - \frac{\sum_{n=1}^{N} \omega_n(k)}{\sum_{n=1}^{N} P_n(k)N} \tag{9.5}$$

9.4.3 Reliability

The metric of **reliability** is crucial in any network system, and in the context of resource allocation in a Cloud RAN system, it is particularly important. Reliability in this setting is related to transmission loss, as defined in Chapter 6. Transmission loss is the ratio of the number of expired, unserved transmission requests to the total number of transmission requests made by all CUs. Therefore, the formula for calculating the system's reliability is as follows:

$$\text{Reliability} = 1 - \frac{\text{Expired transmission requests}}{\text{Total transmission requests}} \tag{9.6}$$

Here, a higher value of "reliability" indicates fewer lost or expired transmission requests, which in turn suggests a more efficient and reliable system.

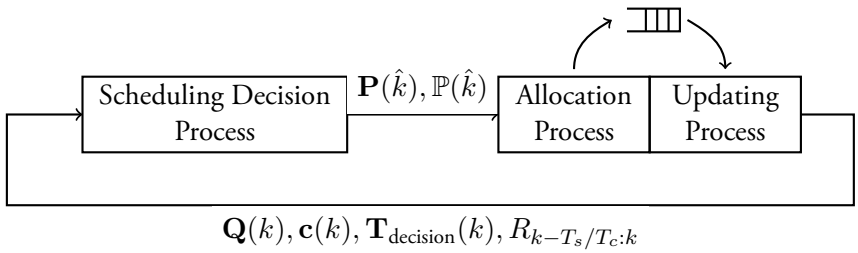


Figure 9.3: Diagram illustrating the Cloud RAN scheduling process during time slot k. The figure shows the sequence of events beginning with the transmission of an update message from the RRH to the BBU pool. An allocation decision based on the received $\mathbf{P}(k)$ is subsequently actuated at the RRH. Simultaneously, the BBU pool formulates a scheduling decision for a future time slot, denoted as $\hat{k}$, and transmits this decision back to the RRH.

9.5 Punctual Cloud for Radio Resource Allocation

Based on the model and the performance metrics defined for the scheduling process, a custom-built scheduler tailored for an industrial scenario in a Cloud RAN system should aim to achieve two primary objectives:

1. Fairly allocate pilots to all CUs with pending transmission requests before these requests expire.
2. Avoiding starvation of the background traffic, which is a consequence of resource waste.

As demonstrated in Chapter 6, a naive scheduling approach can indeed meet the reliability standards required for industrial applications by keeping the loss rate under 5%. However, such a naive method comes at the cost of excessive pilot waste in order to maintain high reliability.

Given the inherently stochastic nature of Cloud RAN, we introduce the **Punctual Cloud** Framework, featuring a novel resource allocation strategy that ensures the timely delivery of scheduling decisions. This framework adeptly manages the delayed and out-of-sequence messages that are characteristic of a stochastic Cloud RAN system. As a result, we observe a significant improvement in pilot utilization without sacrificing reliability—maintaining the loss rate under the 5% threshold. A comprehensive overview of this innovative framework is depicted in Figure 9.3.

Our proposed punctual cloud framework divides the pilot scheduling process over Cloud RAN into three sub-processes:

- The **allocation process** on RRH that allocates the pilots to the CUs with pend-

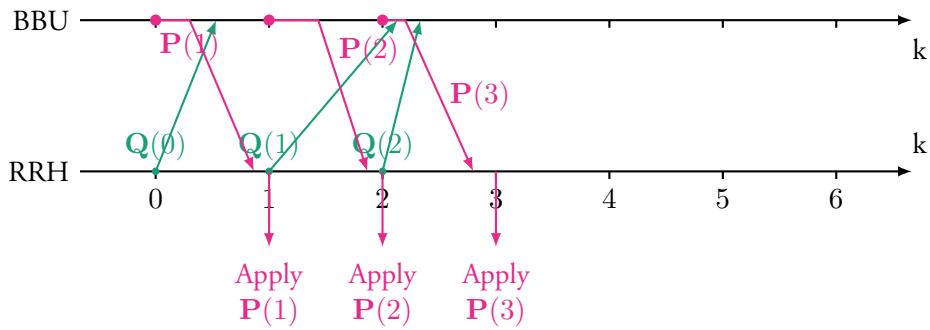


Figure 9.4: Time series plot of subsequent update messages $\mathbf{Q}(k)$ and decision messages $\mathbf{P}(k)$ under the punctual cloud framework.

ing transmissions.

- The **updating process** that sends the updates about the pending transmissions to the BBU pool by the RRH.
- The **scheduling decision process** that makes the scheduling decisions in the BBU pool and sends the decisions to the RRH.

As depicted in Figure 9.4, our punctual cloud framework addresses the challenges of stochastic delays by designing the updating process and scheduling decision process to function asynchronously and independently. In this architecture, the BBU pool periodically generates scheduling decisions for a prospective actuation slot, denoted as $\hat{k}$. These decisions are formulated based on the predicted time of arrival for the decision itself, along with an estimated state of the RRH at that future time, $\hat{k}$.

It is worth noting that the actual state, represented by $\mathbf{Q}(\hat{k})$, may not yet be available at the BBU pool when the decision for $\hat{k}$ is made. To overcome this limitation, our framework relies on a robust state estimation methodology that leverages historical data from previous updates. Additional mechanisms, such as message buffering and redundancy, are also implemented to ensure timely application of scheduling decisions.

In the subsequent sections, we delve into the intricacies of each sub-process, describing how they collectively contribute to robust and efficient resource allocation in our Cloud RAN system.

9.5.1 Updating Process

At each time slot k, the RRH sends an update message to the BBU pool. This message serves the function of enabling state estimation for the RRH within the BBU pool. The constituents of the update message are as follows:

- **Pending Transmission Requests**: Represented by $\mathbf{Q}(k)$, this informs the BBU pool about the number of pending transmission requests for CUs.
- **Inter-Arrival Times**: Denoted by $\mathbf{c}(k) = \{\mathbf{c}_1(k), \mathbf{c}_2(k), ...\mathbf{c}_N(k)\}$, these times capture the intervals between transmission requests from each CU that arrived during the preceding slot $k-1$. Given that the inter-arrival time c_n may be shorter than the length of a time slot T_s, a CU could potentially release multiple transmission requests during that time. Consequently, $\mathbf{c}_n(k)$ is a set comprising all the inter-arrival samples for CU_n as measured during slot $k-1$.
- **Measured Decision Delays**: This array, denoted by $\mathbf{T}_{\text{decision}}(k)$, encapsulates the measured delay samples for decision messages that have arrived during the time slot $k-1$.

Additionally, at regular intervals defined by T_p, the update message from the RRH includes the punctuality metric capturing the efficiency of resource allocation over the last T_p time period. If sent at slot k, this is denoted as $R_{k-T_p/T_s:k}$. For the sake of simplicity—and given that T_p remains constant–we use $R(k) = R_{k-T_p/T_s:k}$ to signify the punctuality value forwarded at slot k. This metric contributes to the horizon prediction of decision arrivals in the scheduling decision process.

9.5.2 Allocation Process

At each slot $\hat{k}$, after the update message has been sent, the RRH executes the allocation process based on the received decision. This process assigns pilots to the active CUs. The intricacies of the allocation process are illustrated in Figure 9.5.

To address the challenges of decisions arriving too early, too late, or out of sequence, we advocate for the use of a decision buffering mechanism at the RRH. In this approach, the RRH stores incoming decisions and only implements them at their specified actuation slots. Should a decision fail to arrive in time for its intended actuation slot, the RRH will instead apply a stored decision, $\mathbf{P}(\hat{k}')$, designated for the time slot $\hat{k}'$, that is closest to $\hat{k}$. This strategy operates under the rationale that the state estimate for the slot closest to the intended one will, on average, provide the next best approximation.

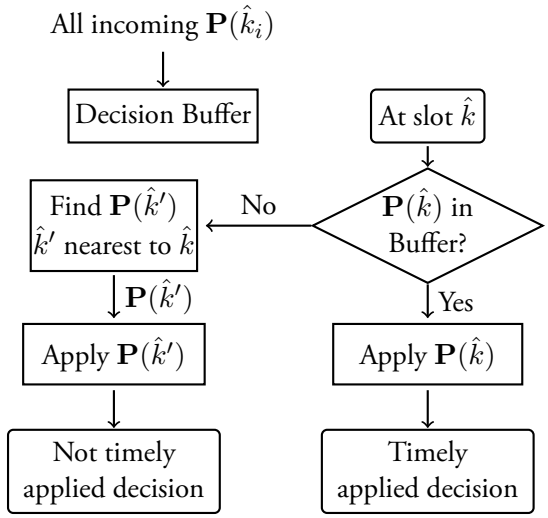


Figure 9.5: Allocation process at RRH.

9.5.3 Scheduling Decision Process

The scheduling decision process within the framework is orchestrated to formulate a decision $\mathbf{P}(\hat{k})$ at the BBU pool, intended for execution at a future time slot $\hat{k}$. To make an effective scheduling decision, the process needs to estimate:

1. The exact time slot $\hat{k}$ at which the scheduling decision will be actuated by the allocation process.
2. The number of pending transmission requests from each CU at time $\hat{k}$, denoted by $\mathbf{Q}(\hat{k})$.

Consequently, the scheduling decision process is segmented into three sequential sub-processes: **arrival estimation**, **horizon prediction**, and **queue estimation**. These sub-processes collectively provide the necessary estimates for making an informed scheduling decision, as depicted in Figure 9.6.

In subsequent sections, we elaborate on each of these sub-processes. It is worth noting that alternative methods could be employed within each sub-process, provided they maintain the prescribed inputs and outputs, as outlined in Figure 9.6.

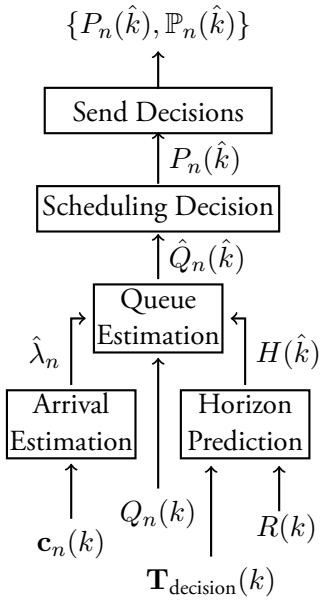


Figure 9.6: Scheduling Decision process in the BBU pool based on the update message sent by the RRH at time k. The process performs a decision expected to be applied by the RRH at slot $\hat{k}$, where $\hat{k} \geq k$

Scheduling Decision

The scheduler formulates a decision, $\mathbf{P}(\hat{k})$, intended for application at a future time slot, $\hat{k}$. This decision is made based on an estimated state of the RRH on all active CUs at slot $\hat{k}$.

While our approach employs a **greedy allocation strategy**, it should be noted that alternative scheduling methodologies can also be utilized. Specifically, the decision for CU_n is set to $P_n(\hat{k}) = 1$ if $Q_n(\hat{k})$ is non-zero and ranks among the p highest values within the set $\mathbf{Q}(\hat{k})$.

The decision message dispatched by the scheduler includes not only the freshly made decision $\mathbf{P}(\hat{k})$, but also h redundant decisions, denoted as $\mathbb{P}(\hat{k}) = \{\mathbf{P}(\hat{k}-1), \mathbf{P}(\hat{k}-2), ...\mathbf{P}(\hat{k}-h)\}$. These redundant decisions are intended for actuation slots preceding $\hat{k}$. The incorporation of redundancy ensures that even if a decision aimed for slot $\hat{k}$ experiences delays—arriving later than its designated actuation slot—subsequent decision messages might still deliver it in a timely manner to the RRH. This mechanism not only enhances the likelihood of timely decision implementation but also optimizes resource utilization, as demonstrated in the subsequent results section.

Queue Estimation

The decision $\mathbf{P}(\hat{k})$ is indented for actuation at a future slot $\hat{k}$. This necessitates an estimation of the system state at that future slot, which is denoted as $\hat{\mathbf{Q}}(\hat{k})$.

For the sake of this estimation, we consider the most recent update message received by the BBU pool, which provides the state $Q_n(k)$ for CU_n. Assuming an average arrival rate of transmission requests for CU_n to be λ_n, and a predicted time horizon $H(k)$ from the current slot k to the future slot $\hat{k}$, the queue sizes $\hat{Q}_n(\hat{k})$ can be estimated, as detailed in Equation (9.7).

$$\begin{aligned} &\hat{Q}_n(\hat{k}) = Q_n(k) + \lambda_n \times (\hat{k} - k) \times T_s - \sum_{\kappa=k}^{\hat{k}-1} P_n(\kappa) \\ &\text{where } \hat{k} = k + H(k) \end{aligned} \tag{9.7}$$

The term $\sum_{\kappa=k}^{\hat{k}-1} P_n(\kappa)$ represents the cumulative decisions anticipated to be executed from the current slot k up to slot $\hat{k} - 1$.

Arrival Process Estimation

In Equation (9.7), the term $\lambda_n \times (\hat{k} - k) \times T_s$ is used to predict the number of transmission requests that have been released by CU_n during k and $\hat{k}$. We use Exponential Moving Average (EMA) to estimate the average inter-arrival time $\hat{c}_n$ of the transmission requests for CU_n, which gives:

$$\hat{c}_n^+ = \alpha_c \hat{c}_n^- + (1 - \alpha_c) c_n \tag{9.8}$$

Here, c_n is taken from the inter-arrival time sample array $\mathbf{c_n}(k)$ reported in the most recent update message from the RRH to the BBU pool.

The term $\hat{c}_n^+$ the updated estimate of c_n, the inter-arrival time of transmission requests from CU_n. On the other hand, $\hat{c}_k^-$ signifies the previous estimate. The weight α_c parameterizes the EMA estimator, determining the balance between the old and new information. Subsequently, the average arrival rate for CU_n can straightforwardly be computed as follows:

$$\hat{\lambda}_n = \frac{1}{\hat{c}_n^+} \tag{9.9}$$

Predicted Time Horizon

Considering a scheduling decision based on an update message sent at time k, intended for application at future slot $\hat{k}$, where $\hat{k} \geq k$, we introduce $H(k) = \hat{k} - k$, as the **predicted time horizon**. This time horizon is vital in our approach as it determines the allowable delay for a decision message. A lengthier time horizon generally improves the likelihood of timely decision application but introduces a trade-off of increased estimation inaccuracy in queue estimation.

To calculate $H(k)$, we propose a formula that incorporates an estimated average decision delay $\hat{T}_{\text{decision}}$ and an additional offset σ:

$$H(\hat{k}) = \left\lceil \frac{\hat{T}^{+}_{\text{decision}}}{T_s} \right\rceil + \sigma^{+} + \left\lceil \frac{T_{\text{update}}(k)}{T_s} \right\rceil \tag{9.10}$$

Here, the time horizon spans from k to $\hat{k}$ and includes both T_{update} and T_{decision}. We solely need to estimate T_{decision} as $T_{\text{update}}(k)$ is a known duration from k to the time when the scheduling decision is made.

The average decision delay is estimated using an EMA estimator with weight α_d:

$$\hat{T}^{+}_{\text{decision}} = \alpha_d \hat{T}^{-}_{\text{decision}} + (1 - \alpha_d) T_{\text{decision}} \tag{9.11}$$

Similar to the average inter-arrival time estimator in Section 9.5.3, T_{decision} is a sample of the measured decision delay $\mathbf{T}_{\text{decision}}(k)$ that is informed in the update message. $\hat{T}^{-}_{\text{decision}}$ is the previous estimate of the average decision delay.

We further propose a step-controller to adjust the offset σ, which operates based on a feedback loop measuring the punctuality $R_(k)$:

$$\sigma^{+} = \begin{cases} \sigma^{-} + 1 & \text{if } R(k) < r \\ \sigma^{-} & \text{Otherwise} \end{cases} \tag{9.12}$$

Here, σ^{-} serves as the previous offset value and is initialized to zero. The value r establishes the lower bound for the punctuality metric. The step-controller, as defined in Equation (9.12), is activated under the condition that the average network delay either shows minor fluctuations or increases. Should the estimated mean delay experience a decrease, the value of σ^{-} is reset to 0, prompting the controller to recalibrate the offset.

For accurate feedback, $R(k)$ should be determined based on a sufficiently large sequence of past slots. We define a distinct sampling time T_p for the step controller that is considerably greater than the length of a scheduling time slot T_s. As a result,

Table 9.1: Parameters of Transmission Arrival Process

Parameter name	Value	Symbol
inter-arrival time mean	10 ms	c
inter-arrival time std.	0.0005	δ
Number of CUs	20	U
Deadline of a transmission from CU_n	10 ms	D_n

$R(k)$ is sampled every T_p/T_s slots. This design allows for the continual estimation of $\hat{T}_{\text{decision}}$ at each T_s, while adjustments to σ is made at the less frequent interval of T_p.

9.6 Simulation

In this section, we describe the simulation we implement for evaluating the performance of the punctual cloud framework over Cloud RAN. Our simulation evaluation focuses on two key performance metrics: punctuality and pilot utilization. We investigate how these metrics are influenced by the inherent stochastic properties of Cloud RAN through simulation.

The simulation environment is implemented using SimPy[1]. Each experiment was executed for a total simulated system time of $T = 200$s, and the presented results are derived from the average of 20 independent runs.

9.6.1 Simulation Parameters

The simulation model necessitates the configuration of various system parameters, which are elaborated upon in the subsequent sections.

Arrival process of transmissions

To emulate traffic patterns relevant to time-critical industrial applications, we employ the industry and IoT traffic models as outlined in [HMH18]. Each CU_n initiates transmissions following a homogeneous periodic stochastic process, where the inter-arrival time c_n is normally distributed as $c_n \sim \mathcal{N}(c, \delta^2)$. All parameters governing the arrival process of these transmissions, along with the specific values employed in our simulations, are summarized in Table 9.1.

[1] `https://simpy.readthedocs.io/en/latest/`

Table 9.2: Parameters of Delay Distributions in the Simulation

Distribution name	CV^2	Probability density function
Deterministic	0	$p(x) = 1,$ when $x = \mu$
Erlang	0.5	$p(x) = (\frac{2}{\mu})^2 x e^{-\frac{2x}{\mu}} m$
Exponential	1	$p(x) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$
Hyper-exponential	2	$p(x) = \frac{1}{2\mu_1} e^{-\frac{x}{\mu_1}} + \frac{1}{2\mu_2} e^{-\frac{x}{\mu_2}}$ $\mu_1 = \mu(1 + \frac{\sqrt{2}}{2}), \mu_2 = \mu(1 - \frac{\sqrt{2}}{2})$

Stochastic delay

To model the cloud delays T_{update} and T_{decision}, we employ the exponential distribution family. For all these distributions, the mean delay value is set to μ. Our simulation aims to explore how varying delay distributions and the mean delay μ impact overall system performance.

In the simulation, we examine the system's behaviour under different cloud delay scenarios: deterministic, Erlang-distributed, Exponentially-distributed, and Hyper-exponentially-distributed. Table 9.2 details the coefficient of variance CV^2 and Probability Density Function (PDF) associated with each distribution. We varied the mean delay, μ, in a range from 0ms to 4ms. Here, $\mu = 0$ms serves as a baseline, representing a scenario where the scheduler is co-located with the RRH.

Scheduling strategy

The parameters governing the allocation process, which is situated in the RRH, are delineated in Table 9.3. These parameter values are aligned with the radio spectrum specifications of our massive MIMO testbed LuMaMi[Mal+17]. Meanwhile, Table 9.4 lists the parameters employed in the scheduling decision process, which is hosted in the BBU pool.

Table 9.3: Parameters of the Allocation Process in the RRH

Parameter name	Value	Symbol
Scheduling time slot length	0.5 ms	T_s
Number of available pilots per slot	12	p
Number of transmission requests served by a pilot	1	-

Table 9.4: Parameters of the Decision Making Process at BBU

Component	Parameter name	Value	Symbol
Arrival Estimation	EMA weight	0.999	α_c
Horizon Prediction	EMA weight	0.999	α_d
	Lower bound reference	90%	r
	Sampling time	2000ms	T_p
Redundant Decisions	No. of redundancy	2	h

9.6.2 Evaluation Methods

The primary goal of our evaluation is to demonstrate the efficacy of the punctual cloud framework in counteracting the adverse effects of the Cloud RAN's inherent stochastic properties. Specifically, we aim to show that our framework enhances pilot utilization without sacrificing the system's reliability. To accomplish this, we focus on the framework's ability to address the challenges posed by delayed and out-of-sequence decision messages. In the results section, we compare the performance of our punctual cloud framework against three alternative methods that lack a comprehensive set of mitigation strategies. These comparative methods and their associated system parameters are detailed in Table 9.5. For reference, the **Naive Scheduling** method is similar to the one we employed for evaluating the Cloud RAN system in Chapter 6.

As previously outlined in Chapter 6, we have thoroughly examined the reliability performance of the Cloud RAN system through simulation. Our findings confirm that the system can sustain an availability rate exceeding 95%, aligning with industrial standards as stipulated in [Gro19]. Therefore, our current simulation does not focus on evaluating the system's reliability performance.

For the present set of experiments, we adhere to the 95% availability criteria as a baseline requirement, setting the maximum tolerable transmission loss at 5%. It's worth noting that, unless specifically mentioned, all performance metrics assessed in this study adhere to this minimum availability threshold of 95%.

Table 9.5: Evaluated Scheduling Strategies in the Experiments

Method Name	Parameters
Punctual Cloud	Indicated in Table 9.4
Single Decision	Same as Punctual Cloud but $h = 0$
Short Horizon	Same as Punctual Cloud but $h = 0, \sigma \equiv 0$
Naive Scheduling	Described in Section 9.3

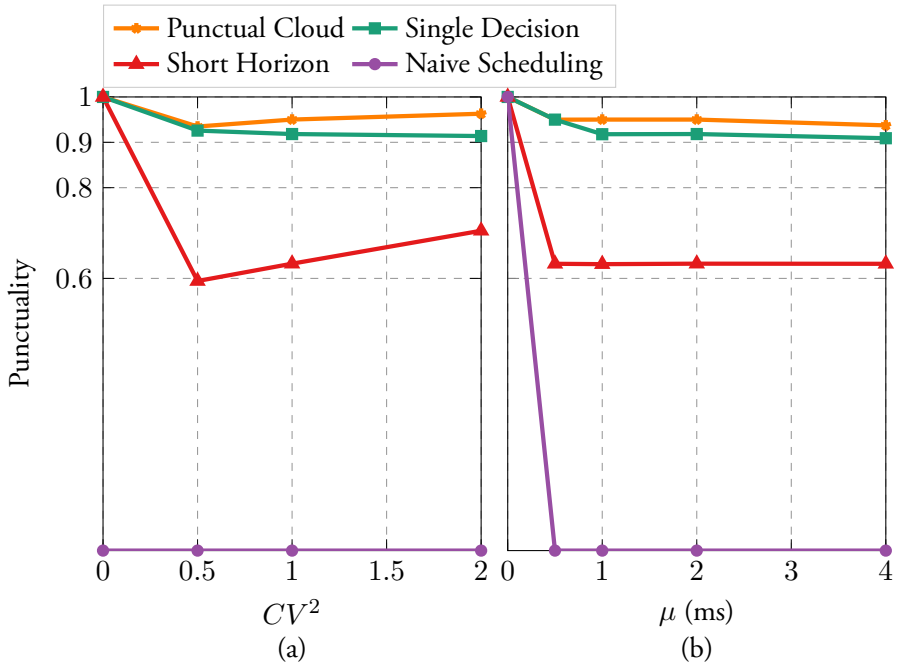


Figure 9.7: Timely applied decision for the four methods (a) under different delay distributions when μ=2ms and (b) when μ increases for exponentially distributed delays.

9.7 Simulation Results

In this section, we delve into the results of our simulation to demonstrate the efficacy of our proposed scheduling framework. Specifically, we focus on how our strategy enhances pilot utilization rates while still fulfilling the industrial reliability standards as outlined in [Gro19]. We first examine the outcome of our approach in terms of the timely application of scheduling decisions. Subsequently, we explore how these timely decisions influence the efficiency of the pilot scheduling process. Importantly, all reported confidence intervals lie within a 10% range of the respective mean values.

9.7.1 Punctuality

Leveraging better punctuality performance, our scheduling strategy effectively counters the challenges presented by a Cloud RAN system. As a baseline for comparison, Figure 9.7 shows that with the Naive Scheduling method, no decisions are applied in a timely manner. This is primarily because the naive scheduler fails to consider the cloud delays, causing all decisions to arrive later than their intended actuation slots.

As depicted in Figure 9.7, extending the prediction horizon improves punctuality. This improvement is evident when comparing the Naive Scheduling, Short Horizon, and Single Decision methods, where the length of the prediction horizon successively increases. Intuitively, a longer prediction horizon allows decisions to arrive earlier than the intended actuation time, thus increasing their chances of being timely applied. Furthermore, the complete punctual cloud framework includes redundant messages, shows a higher punctuality rate across all experiments.

Interestingly, Figure 9.7 also demonstrates that the average delay length does not significantly impact the punctuality performance. This stability is largely due to our estimation of average decision delays. Moreover, a greater variance in the delay distribution may actually enhance the punctuality. This intriguing finding mainly results from the varying distributions we utilized to simulate cloud delays. Specifically, for certain hyper-exponential distributions, the probability of $d \leq \mu$ exceeds that in other distributions.

In summary, punctuality performance is intricately linked to both delay distribution and the prediction horizon. In this problem, we leverage the strategy outlined in Section 9.5.3 to determine the prediction horizon. However, this remains an open question, and alternative methods for prediction could also be effectively employed.

9.7.2 Pilot Utilization

Figure 9.8 illustrates the average pilot utilization achieved by our proposed uplink pilot scheduling strategy, in contrast to that obtained with Naive Scheduling. It is important to note that both the Punctual Cloud and the Naive Scheduling method are able to maintain transmission losses below the 5% threshold mandated by industrial standards. The Naive Scheduling approach adheres to these standards by repeatedly assigning redundant pilots to serve a single transmission. While the Single Decision and Short Horizon methods do enhance the punctuality performance compared to Naive Scheduling, they fall short of meeting industrial standards. This shortfall is because the increase in punctuality is insufficient to compensate inaccuracies resulting from the extended prediction horizon.

Compared to Naive Scheduling, Punctual Cloud boosts pilot utilization significantly, from less than 20% to over 90%. This demonstrates effective mitigation of stochastic delays and out-of-order messages, the primary challenges associated with a Cloud RAN system. When fewer pilots are wasted on the CUs, the system gains greater capacity to accommodate traffic from non-CUs, thereby preventing the starvation of these applications.

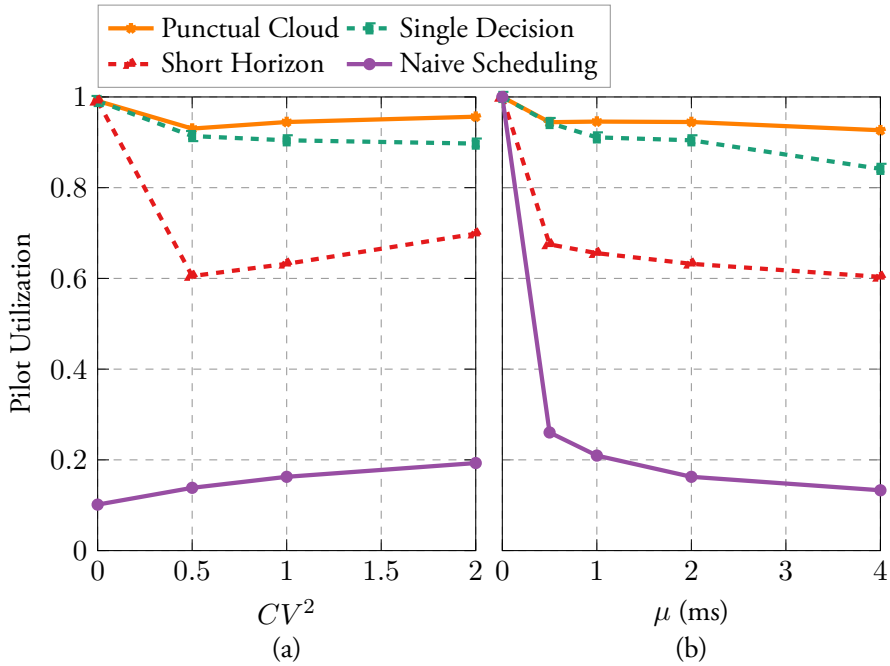


Figure 9.8: Pilot utilization (a) under different delay distributions when μ=2ms and (b) as μ increases for exponentially distributed delay. Performances in dashed lines indicate that the methods didn't meet the reliability requirements.

Upon comparing Figure 9.8 with Figure 9.7, it becomes evident that pilot utilization is closely tied to the punctuality performance. In essence, the more decisions that are timely applied, the fewer pilots are wasted. However, it is also clear that pilot utilization is not solely determined by punctuality; the delays still play a significant role. Longer delays necessitate longer prediction horizons, introducing greater potential for inaccuracies in the queue estimation at the RRH.

9.8 Implementation

We now turn our attention to the practical implementation of our proposed punctual cloud framework. This is deployed on our lab Kubernetes cluster, comprising seven nodes running Ubuntu 20.04. The Cloud RAN system is architected as a client-server model, as illustrated in Figure 9.9. This setup aligns with our earlier implementation of the CCS, depicted in Figure 6.7.

It is worth noting that the predominant architectural style for current cloud and network applications is still RESTful, largely due to its proven maturity and stability

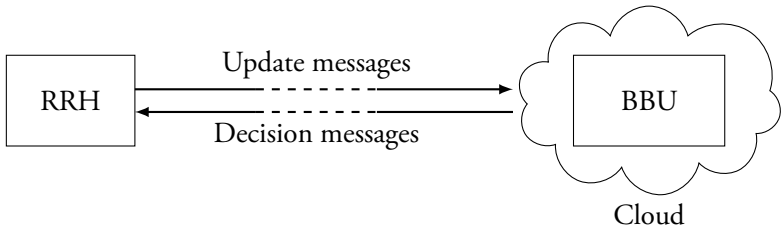


Figure 9.9: System components in a general deployment of Cloud RAN.

[Diz+19]. To align with this industry norm, our implementation of the punctual cloud framework leverages standard HTTP1.1 for deployment. As we will demonstrate in the subsequent evaluation section, the operational latencies introduced by transport and application-level protocols have a negligible impact on system performance within our deployment model.

9.8.1 Punctual Cloud as Kubernetes Services

As previously elaborated, the resource allocation for the Cloud RAN system involves three interconnected processes. The update and allocation processes are executed on the RRH, which acts as the client sending periodic update messages to the BBU pool. The BBU is hosted in our lab Kubernetes edge cluster and runs the scheduling decision process as a cloud service.

In our practical setup, the update and allocation processes remain in simulation on an Ubuntu desktop but transmit real-time update messages with a period T_s. Here, we focus on the cloud-based scheduling decision process, consisting of three subprocesses: arrival process estimation, queue estimation, and scheduling decision.

The scheduling decision process is adapted into two services, as shown in Figure 9.10. The **estimation service** handles arrival process and queue estimations, while the **scheduling service** executes decisions based on these estimates.

Both services are containerized and deployed in scalable Pods within the Kubernetes cluster. Update messages from the RRH are transmitted via HTTP requests. These are load-balanced by an Nginx[2] ingress controller and directed towards the scheduling service. This service, in turn, passes state information to the estimation service through another HTTP request. The estimation service computes and returns the prediction horizon and estimated state $\hat{\mathbf{Q}}(\hat{k})$, enabling the scheduling service to generate and return the corresponding decision $\mathbf{P}(\hat{k})$.

[2] `https://www.nginx.com/products/nginx-ingress-controller/`

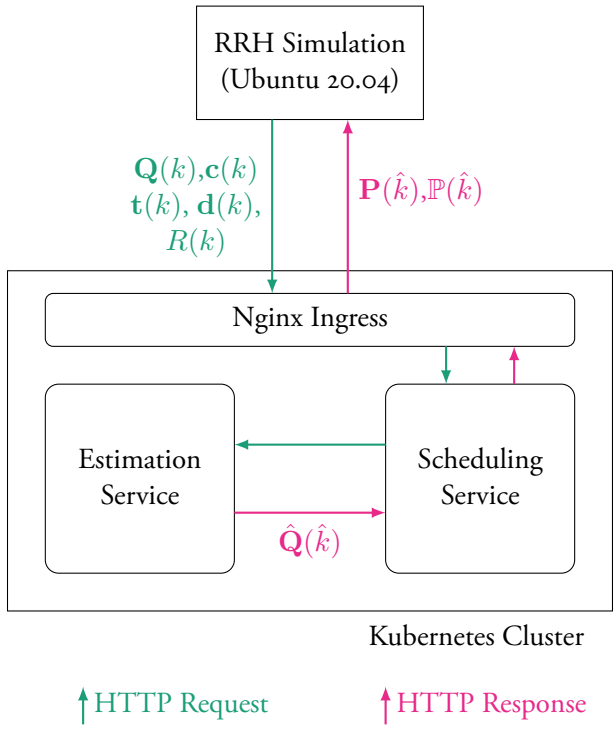


Figure 9.10: Scheduling decision process as cloud application in Kubernetes

It's worth noting that while we have used an Nginx ingress controller for load balancing, alternative configurations like NodePort are also viable. The choice of ingress depends on the specific cloud environment and may be replaced by other load balancing solutions supported by the cloud provider.

Our modular architecture allows for easy adaptability to various real-time applications. The scheduling service remains consistent with naive scheduling and only communicates with the added estimation service. Additionally, while we've employed HTTP for inter-service communication, adding a service mesh layer like Istio[3] or using other protocols like gRPC[4] is straightforward.

The full code for both implementation examples—with and without a service mesh—is available on GitHub[5].

[3] `https://istio.io`
[4] `https://grpc.io`
[5] `https://github.com/HaoruiPeng/cloud-scheduler-mesh`

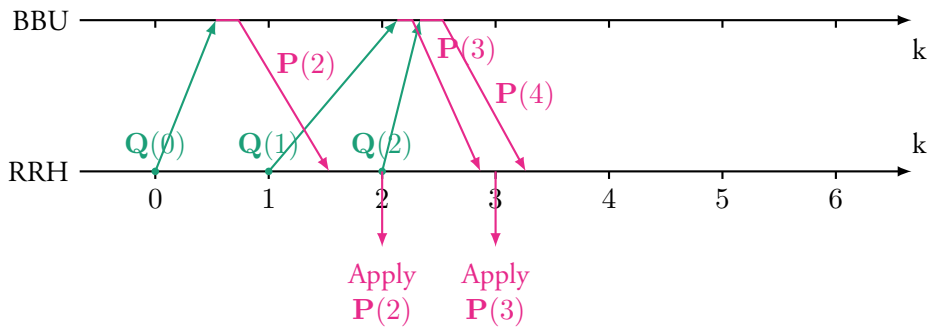


Figure 9.11: Time series pilot of subsequent update messages $\mathbf{Q}(k)$ and decision messages $\mathbf{P}(k)$ in the implemented punctual cloud framework.

9.8.2 Implementation details and deviations from simulation

It's imperative to mention that the methods employed in the actual implementation differ slightly from those used in simulation, primarily due to the practical constraints of system synchronization and cloud delay measurement.

In our implementation, the RRH and BBU pool are not perfectly time-synchronized, which prevents accurate measurement of cloud delays T_{update} and T_{decision} without additional synchronization mechanisms. Consequently, the scheduling decision process at the BBU pool is initiated upon receipt of an update message from the RRH, rather than operating independently. This behaviour is graphically represented in Figure 9.11.

In the **Horizon Prediction** sub-process, the simulation assumed measurable cloud delays to estimate the decision delay T_{decision}. However, given the time synchronization issue in the actual implementation, we estimate response delays T_r to predict the time horizon for the arrival of each decision message.

Significant differences are also present in the **Queue Estimation** sub-process. Practically speaking, each update message usually has either one or zero pending transmission requests from each user. To improve estimation accuracy and reduce pilot waste, we use a specific criterion. Namely, we check for the existence of an integer i, that satisfies Equation (9.13), where $t_n(k)$ is the timestamp for the most recent transmission request from each user CU_n, and $\mathbf{t}(k) = \{t_n(k)\}$ for $n \in \{1, 2, ...N\}$.

$$(i-1)T_s < t_n(k) + i\hat{c}_n \leq kT_s \tag{9.13}$$

If such an integer i exits, it is highly likely that CU_n will initiate a new transmission request at the future time $\hat{k}$, allowing us to estimate the queue state $\hat{Q}_n(\hat{k})$ as 1. Otherwise, the queue state is estimated as 0.

Algorithm 1: Estimation on $\hat{\mathbf{Q}}(\hat{k})$ and generating scheduling decision $\mathbf{P}(\hat{k})$ in the cloud

Data: $\alpha_c, \alpha_d, r, \hat{T_r}, \{\hat{c}_n\}, \{\hat{t}_n\}, \sigma = 0, T_s$
Input: $\mathbf{Q}(k), \mathbf{T_r}(k), R(k), \{t_n(k)\}, \{c_n(k)\}$
`/* Estimation service: Horizon Prediction */`
foreach $T_r \in \mathbf{T_r}(k)$ **do**
  $\hat{T_r} \leftarrow \alpha_d \hat{T_r} + (1 - \alpha_d) T_r$;
end
if $R(k) \leq r$ **then**
  $\sigma \leftarrow \sigma + 1$;
end
$\hat{k} \leftarrow k + \lceil \frac{\hat{T_r}}{T_s} \rceil + \sigma$;
`/* Estimation service: arrival estimation and queue estimation */`
foreach $n \in \{1, \ldots, N\}$ **do**
  $\hat{c}_n \leftarrow \alpha_c \hat{c}_n + (1 - \alpha_c) c_n(k)$;
  $i \leftarrow 0$;
  $\hat{Q}_n(\hat{k}) \leftarrow 0$
  while $t_n(k) + i\hat{c}_n \leq kT_s$ **do**
    if $(k-1)T_s < t_n(k) + i\hat{c}_n \leq kT_s$ **then**
      $\hat{Q}_n(\hat{k}) = 1$;
      break;
    end
  end
end
$\hat{\mathbf{Q}}(\hat{k}) \leftarrow \{\hat{Q}_1(\hat{k}), \hat{Q}_2(\hat{k}) \ldots \hat{Q}_N(\hat{k})\}$;
`/* Scheduling service */`
$\mathbf{P}(\hat{k}) \leftarrow \mathrm{F}(\hat{\mathbf{Q}}(\hat{k}))$;
Output: $\mathbf{P}(\hat{k})$

Therefore, the update message from the RRH includes not only $\mathbf{Q}(k)$ but also $\mathbf{t}(k)$ to enhance queue estimation accuracy. A comprehensive algorithm for each process within the cloud application is detailed in Algorithm 1.

9.9 Experiment with Punctual Cloud Implementation

Now we describe our experiment setup, designed to put our punctual cloud framework implementation for Cloud RAN resource allocation to the test. Our experiments aim to validate the efficacy of our proposed punctual cloud architecture by contrasting it with a baseline system that employs naive scheduling, as discussed in Section 9.3.

Similar to our simulation setup, the client Ubuntu machine acting as the RRH simulates traffic from N CUs, each generating transmission requests at a periodic rate of c_n. These requests are then processed by a cloud-based scheduling service within the Kubernetes cluster.

Connectivity between the RRH and the Kubernetes cluster housing the BBU pool is facilitated by a 1Gbps Ethernet connection. ICMP "PING" measurements indicate a negligible network latency of 0.215ms between the client and the master node of the cluster, which is negligible compared to the application's operational frequency.

To emulate network conditions similar to WANs, we deploy Netem[6] on the client machine, controlling delay mean and jitter values according to a Pareto distribution. The mean delay value μ ranges from 0 ms to $0.8T_s$, while the jitter value σ varies up to μ.

It should be highlighted that although delays introduced by Netem are set to be smaller than the T_s interval, the actual response delays can be notably longer due to TCP operations, admission time, and internal cluster routing. Notably, our punctual cloud framework yields longer average response delays than the single service naive scheduling baseline, attributed to the inter-service communications within the cluster.

Table 9.6 provides a detailed rundown of the parameters utilized in our experiments. Due to the hardware limitations inherent to general-purpose computing infrastructures, we were unable to achieve the high-frequency resource allocation that was possible in the simulation. Each parameter set was tested ten times, and each test iteration ran for a duration of 12.5 minutes.

Performance metrics for our resource allocation application are assessed based on the criteria outlined in Section 9.3, focusing on **punctuality**, **pilot utilization**, and **reliability**.

[6] `https://wiki.linuxfoundation.org/networking/netem`

Table 9.6: Experiment parameters used in the evaluation

Symbol	Parameter Definition	Value
T_s	Scheduling time slot length	50ms
μ	Netem delay configuration value	-
δ	Netem jitter configuration value	-
N	Number of user having resource demands	10
c_n	Transmission requests inter-arrival time of each CU_n	70ms
P	Number of available pilots per slot	12

9.10 Experiment Results

In this section, we delve into the results of our experiments, revealing the effectiveness of our punctual cloud architecture for Cloud RAN resource allocation. As showcased in Figure 9.12 and Figure 9.13, our approach notably enhances application performance across the three key metrics we have identified: punctuality, pilot utilization, and reliability. Importantly, these results are achieved even when faced with extended and unpredictable delays. For consistency, all figures in this section reflect the same Netem configuration.

In the delay evaluation, Netem's mean delay μ was adjusted between 0 and 40ms, and the jitter δ was set at 0.6μ. This implies a coefficient of variance of 0.6 for the delay in the Netem setup.

For the experiments that examined the impact of varying jitter, we fixed Netem's mean delay at $\mu = 25$ms and let σ range from 0 to μ. Under these conditions, the actual mean response delay escalated with an increase in jitter, while maintaining a constant mean delay value. The other parameters for these experiments are detailed in Table 9.6.

It's important to note that the standard deviations associated with our results were negligible, thus we have opted not to display them in the figures.

9.10.1 Punctuality

With naive scheduling, if the response delay exceeds a single scheduling time slot T_s, the corresponding response is considered unpunctual. This leads to the expiration of the queue state before the allocation process can serve it. On the other hand, in a system deployed using our punctual cloud framework, the returned scheduling decision is often aimed at future states. These states are communicated through update messages that may not yet have been generated. Therefore, a response is considered punctual as long as it arrives before the expiration of the state it is designed to serve.

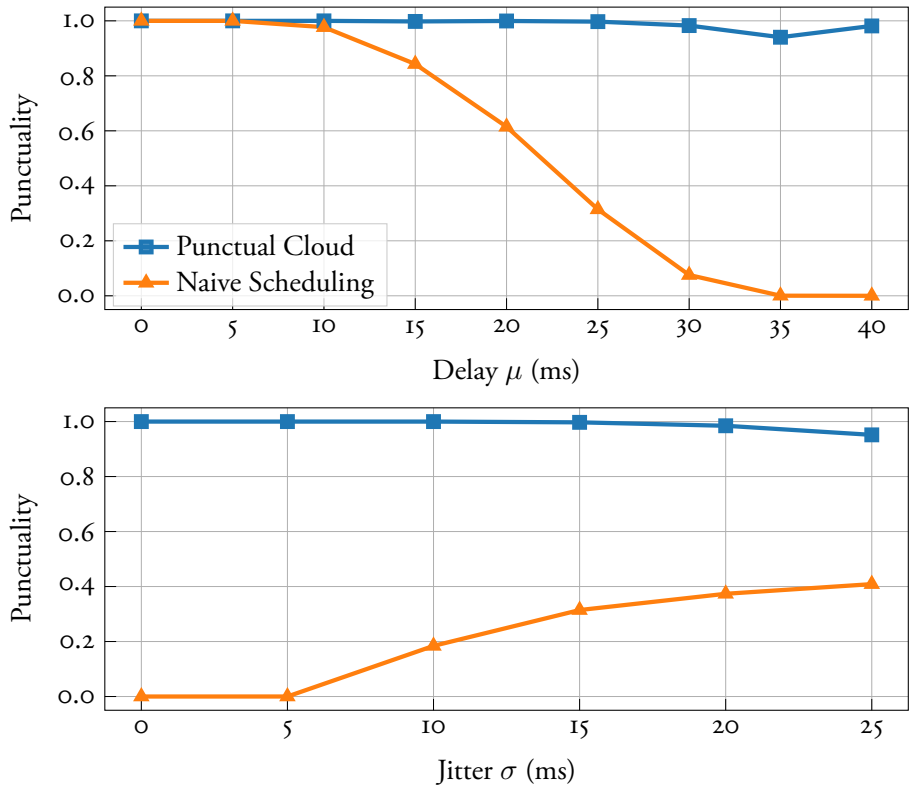


Figure 9.12: Punctuality performance among all the scheduling decisions for experiments under given Netem configurations.

Figure 9.12 illustrates an intriguing behaviour: as jitter in the delay increased, the naive scheduling method exhibited improved punctuality, a trend not mirrored in the punctual cloud performance. This phenomenon can be attributed to the interplay between the empirical cumulative distribution function of the response delay measurements T_r and scheduling time slot T_s. To illustrate, when the jitter value $\delta = 0.4\mu$ and $\mu = 0.5T_s = 25$ms under the Netem configuration, the empirical cumulative probability of $T_r \leq T_s$ was found to be 24.7%. This value jumps to 40.5% when $\delta = 0.8\mu$. This accounts for the increased punctuality performance with greater jitter in naive scheduling. Importantly, our proposed architecture maintains more robust performance in punctuality as either the delay mean μ or the jitter σ increase.

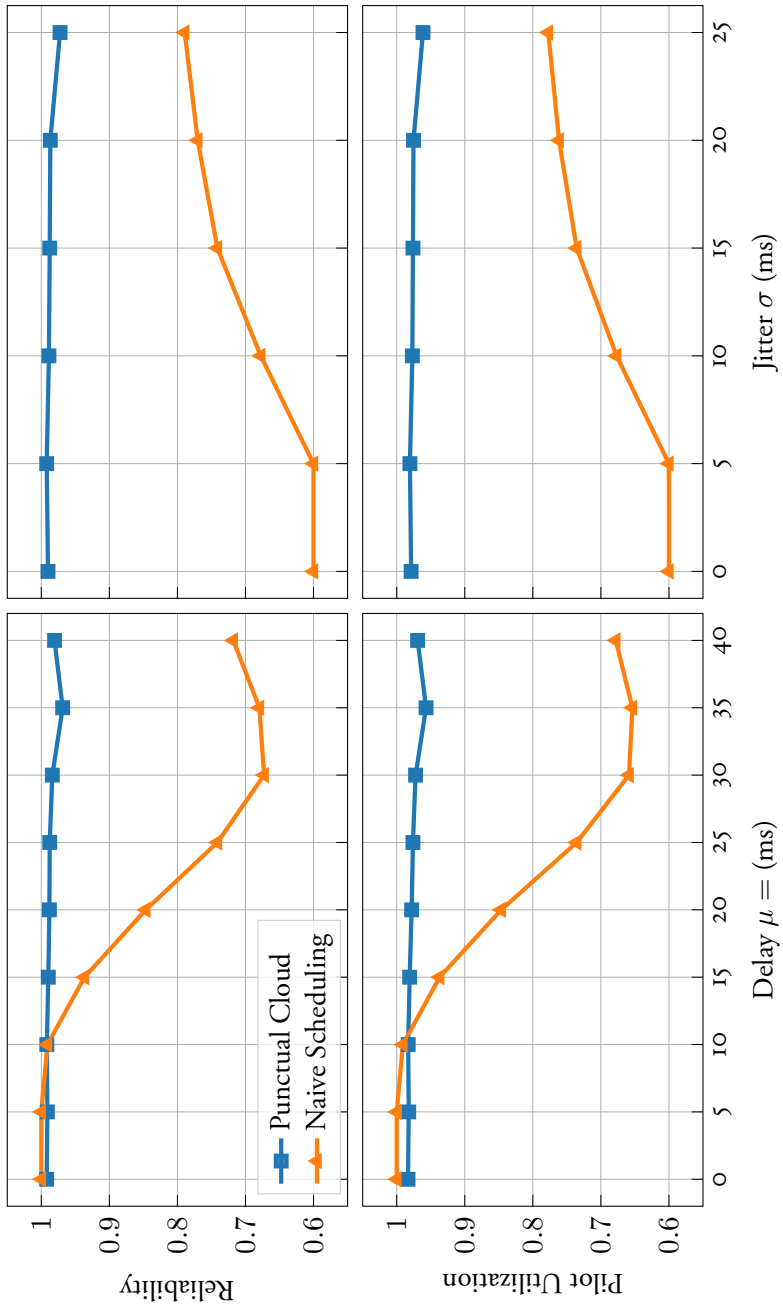


Figure 9.13: Demand failures and resource waste of the resource allocation under given Netem configurations.

9.10.2 Pilot utilization and Reliability

Regarding the resource allocation performance, allocation processes must still conduct scheduling decisions even if the expected decisions are not punctually delivered. In such scenarios, these decisions rely on outdated transmission request queue states, which may not align with current requirements. This misalignment can result in both failures and resource wastes due to false allocations. Consequently, lower punctuality performance is likely to adversely affect other performance metrics. Figure 9.13 depicts the performance in terms of reliability and pilot utilization for two different methods. As illustrated, when the mean delay value μ is minimal and the response delays T_r are shorter than the request interval T_s, there is negligible difference in reliability and utilization between the two approaches. In fact, the naive scheduling outperforms the punctual cloud framework when response delays are sufficiently short to allow most decisions to be delivered punctually, even without additional estimations. This suggests that the efficacy of resource allocation is also contingent on the accuracy of the estimations. However, as the delay lengthens, the estimations within the punctual cloud framework become increasingly beneficial, mitigating the impact of prolonged delays.

As the jitter in the delay increases, Figure 9.13 reveals that our punctual cloud framework performs markedly better, consistent with the higher punctuality depicted in Figure 9.12. Unlike naive scheduling, the performance of the punctual cloud framework is less affected by increasing jitters because its punctuality is not solely dependent on the distribution of response delays.

Lastly, it's worth mentioning that even if most decisions in naive scheduling arrive late, the overall allocation performance doesn't entirely deteriorate. Executing an outdated decision does not invariably result in incorrect allocations. For instance, if the inter-arrival time c_n for a CU_n is longer, the queue state of CUs become less dynamic, thereby reducing the likelihood of resource waste.

9.11 Conclusion on Punctual Cloud for Cloud RAN

This chapter presents our novel punctual cloud framework, specifically designed for addressing the radio resource allocation challenges in Cloud RAN. In this setup, a BBU pool located in a cloud environment is responsible for computing scheduling decisions. These decisions target the RRH massive MIMO uplink pilots for active industrial units in a factory automation scenario. The framework is engineered to cope with the stochastic variables introduced by Cloud RAN, thus ensuring that each scheduling decision is executed in a timely fashion by the allocation processes at the

RRH.

Both simulation and real-world test-bed evaluations—conducted on an edge Kubernetes cluster—demonstrate the effectiveness of our proposed framework. Notably, it mitigates the unpredictabilities inherent in Cloud RAN, such as stochastic delays and out-of-order decision messages.

Within the punctual cloud framework, the decision scheduling processes are encapsulated as a cloud service. This service calculates and delivers scheduling decisions based on estimated response delays and the state of the transmission request queue at the RRH. Our evaluations confirm that the framework succeeds in providing more punctual scheduling decisions. This elevated level of punctuality, in turn, enhances the overall performance of the resource allocation process, yielding higher rates of reliability and pilot utilization.

Chapter 10

Punctual Cloud for Cloud Control Systems

10.1 Targeted system

In Chapter 9, we demonstrated how our punctual cloud framework can effectively mitigate the impact of cloud-induced delays in resource allocation within a Cloud RAN system. Building on that foundation, this chapter shifts the spotlight to Cloud Control Systems (CCSs). As detailed in Part III and depicted in Figure 6.7, a CCS is a cyber-physical system where the controller is deployed as a cloud service, either at the network edge or in a centralized cloud environment. In this system, the control plant transmits its state information to a remote cloud service, which, in turn, sends back the necessary control signals to the plant.

In this chapter, we will extend the applicability of our punctual cloud framework to demonstrate its capacity for mitigating cloud-induced delays across a broader range of cloud applications. Specifically, for a CCS, our framework ensures the prompt delivery of control signals computed in the cloud. The punctual cloud framework thereby enables even a basic controller to handle dynamic, time-sensitive processes over the cloud. We will also present an implementation of this framework for CCS within a microservice architecture, streamlining both the deployment and maintenance of the application.

10.1.1 Control system model

The primary goal of the punctual cloud framework is to enable the deployment of controllers as cloud-based applications. In this section, we define the control system model that serves as the basis for designing our punctual cloud framework.

We focus on a dynamic and observable control system characterized by a nonlinear discrete physical model, as expressed in Equation (10.1). Here, the plant state at time k is denoted as $x(k)$, and the control signal actuated on the plant at that instant is $u(k)$.

$$x(k+1) = f(x(k), u(k)) \tag{10.1}$$

We also present the linearized discrete state-space representation of the control system in Equation (10.2). The system operates with a sampling interval T_s.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{10.2}$$

In the CCS architecture, where the plant is dynamic, the controller operates remotely as a cloud service. This controller generates control signals based on the plant states, which are transmitted over a network. We define the system's **response delay** as T_r, as illustrated in Equation (10.3). This T_r is a stochastic variable with a mean value of μ and a jitter value of δ. It comprises the sum of T_{up}, T_{down} and T_{exe}. Here, T_{up} represents the time required to send state information from the plant to the controller; is the time taken to transmit the control signal from the controller to the plant, and T_{exe} is the execution time to compute the control signal in the cloud. We denote the **discrete response delay** as d, calculated as Equation (10.3).

$$\begin{aligned} T_r &= T_{\text{up}} + T_{\text{down}} + T_{\text{exe}} \\ d &= \lfloor \frac{T_r}{T_s} \rfloor \end{aligned} \tag{10.3}$$

If the discrete response delay $d \geqslant 1$, the cloud control system becomes a time-delay control system with d steps delay in discrete time, and the system can be modelled as Equation (10.4). The top figure of Figure 10.1 illustrates the control diagram of the time-delay system.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k-d) \\ y(k) &= Cx(k) + Du(k-d) \end{aligned} \tag{10.4}$$

10.1.2 Problem definition

As illustrated in the bottom figure of Figure 10.1, in a CCS, when the response delay $d \geqslant 1$, the control signal $u(k)$–computed in the cloud based on $x(k)$–is only received

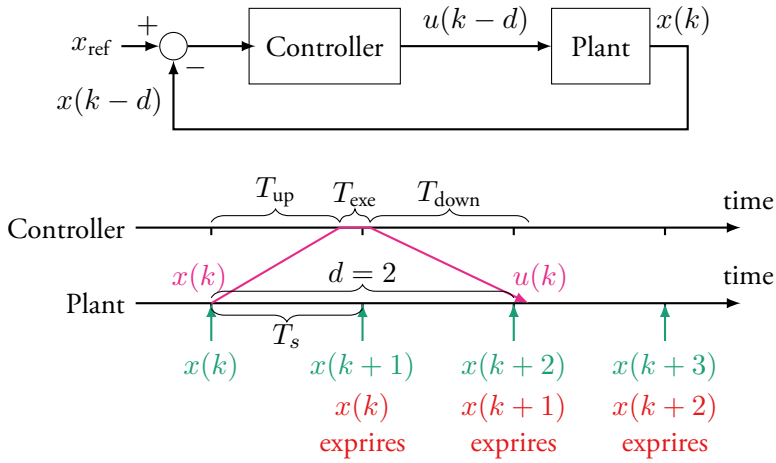


Figure 10.1: Top figure: Control diagram of a time-delay system with d steps discrete response delay; Bottom figure: Time series example when $d = 2$ in CCS.

by the plant when the state has transitioned to $x(k+d)$, and $x(k)$ has expired at that time.

For controllers designed on a linear and delay-free model such as Equation (10.2), functionality is not necessarily compromised. In essence, $u(k)$ may still be applicable to $x(k+d)$ without jeopardizing the plant's stability, especially if the plant operates under slow dynamics and the state $x(k)$ does not significantly deviate from $x(k+d)$. However, this adequacy breaks down when network-induced delays are substantial, specifically when they exceed multiple sampling intervals T_s and the plant dynamics are fast. In these cases, the controller becomes ineffective because $u(k)$ was computed based on an outdated state $x(k)$, which may be considerably different from the current state $x(k+d)$.

Traditional time-delay compensating approaches, such as the Smith Predictor and its variants, have been in use for years [Smi57; Zho06]. MPC, a form of optimal control that employs models to predict and sequence future control signals [Ric+78; 07], has been successfully adapted to counter delays in such systems. This is especially relevant for CCS due to the cloud's high computational capabilities [Ska+19; Årz+18]. However, it is crucial to acknowledge the computational burden associated with MPC, as well as its limitations in handling extended delays, which will be shown later in our evaluation section.

In contrast, our punctual cloud framework offers a more resource-efficient alternative. It uses a simple delay-free controller without requiring alterations to the existing control algorithm. This results in considerably lower computational overhead compared

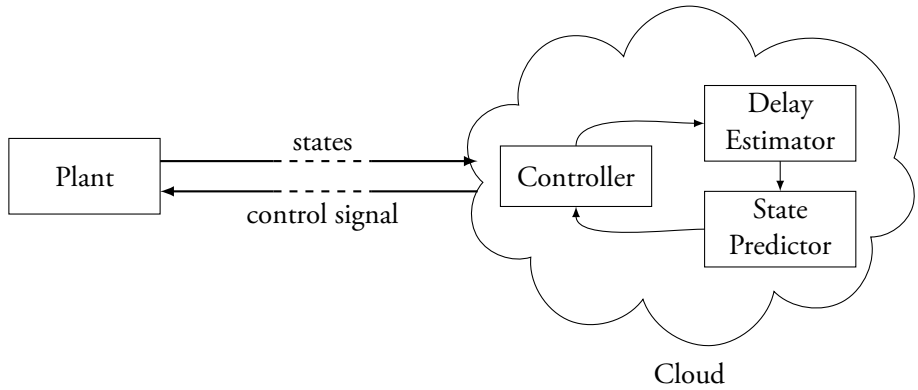


Figure 10.2: Punctual cloud architecture

to MPC, while achieving similar or even superior control performance under certain conditions. The punctual cloud framework ensures that the control signal $u(k)$ is timely applied at the current state $x(k)$, despite substantial network and cloud-induced delays d between the cloud-based controller and the plant.

10.2 Punctual Cloud for CCS

In this section, we present our punctual cloud framework tailored for CCSs. This approach employs a straightforward controller based on a linear, delay-free model to remotely manage the plant, even when faced with significant system-induced delays. Our innovative strategy incorporates three key cloud-based services: a **delay estimator**, a **state predictor**, and a **controller** Service. These services are deployed as a microservice architecture, either in a centralized cloud or at the edge, as depicted in Figure 10.2.

The control diagram under punctual cloud framework is delineated in Figure 10.3. As highlighted in the lower section of the same figure, our state predictor forecasts the future plant state $x(k+d)$ when the cloud-based controller receives the plant state $x(k)$. This enables the controller to generate a future-actuated control signal $u(k+d)$ based on a delay-free model. Unlike MPC, which computes a sequence of states for multiple future time steps, our approach focuses solely on predicting the plant state at a specific future time $k+d$, which coincides with when the control signal will be actuated on the plant.

It's worth noting that the delay estimator, the state predictor, and the controller service are algorithmically independent entities. This modularity means that the algorithm employed in each component can be altered without affecting the others, provided

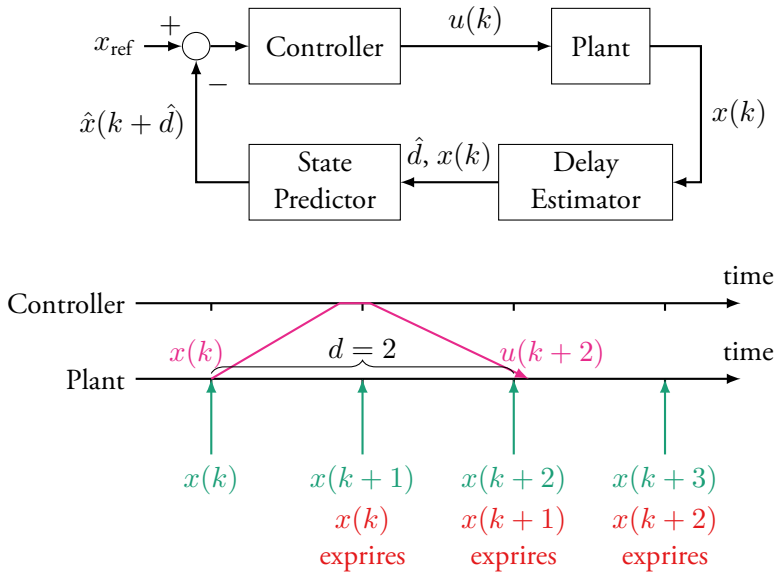


Figure 10.3: Top figure: Control diagram of the punctual cloud for a CCS with d steps discrete response delay; Bottom figure: Time series example when $d = 2$ in CCS with punctual cloud.

that they meet the required input and output specifications as laid out in Figure 10.3. In subsequent sections, we elaborate on the algorithms that we used for each of these three critical components, designed to control a dynamic plant operating at a sampling rate of T_s and capable of handling arbitrary network delays.

10.2.1 Delay estimator

The output of the delay estimator is a current estimation of the discrete response delay $\hat{d}$, which enables the state predictor to forecast $\hat{x}(k+\hat{d})$. Consequently, the controller can then formulate a control signal $u(k+\hat{d})$ based on this prediction, which is scheduled to be actuated on the plant at time $k+\hat{d}$.

Similar to the methodology in Chapter 9, we employ the EMA to estimate the continuous response delay $\hat{T}_r$ for each control signal. This is calculated as Equation (10.5), where ω represents the weight parameter for the EMA estimator, and T_r^{input} is transmitted along with $x(k)$ by the plant, informing the cloud-based services about the response delay incurred by previously delivered control signals.

$$\hat{T}_r = \omega \hat{T}_r + (1-\omega) T_r^{\text{input}} \tag{10.5}$$

Based on this continuous delay estimate, the delay estimator then calculate the discrete

time horizon $\hat{d}$ as follows:

$$\hat{d} = \lfloor \frac{\hat{T}_r}{T_s} \rfloor \tag{10.6}$$

This discrete delay $\hat{d}$ serves as an essential input to the state predictor and controller service, ensuring that control signals are appropriately timed and thereby mitigating the negative impact of network delays on system performance.

10.2.2 State Predictor

Taking the delay estimation $\hat{d}$, the state predictor forecasts a future state $\hat{x}(k+\hat{d})$ based on the current input state $x(k)$. This enables the controller to subsequently generate a control signal $u(x + \hat{d})$ that aligns with this predicted state. In our framework, a Smith Predictor serves as the state predictor component, incorporating a nonlinear system model for improved accuracy.

Smith Predictors operate on the principle that if the system model accurately mimics the plant, a controller for a time-delayed system can be designed as if it were delay-free [Zho06]. However, since a model is never a perfect representation of the actual plant, Smith Predictors also account for this discrepancy. They take into account the error in the previous state prediction $\hat{x}(k)$ and applies it to refine the next state prediction $\hat{x}(k+1)$. Hence, the resulting control signal is based on $\hat{x}(k+1) + x(k) - \hat{x}(k)$.

To be more specific, the state predictor leverages the nonlinear model of the plant to reduce model error compared to a linear approximation. The one-step-ahead prediction of the plant state in discrete time is given by Equation (10.7):

$$\begin{aligned} e(k) &= x(k) - \hat{x}(k) \\ \hat{x}(k+1) &= f(x(k), u(k)) + e(k) \end{aligned} \tag{10.7}$$

To predict the state $\hat{d}$ steps into the future, Equation (10.7) is iteratively run for $\hat{d}$ times, as detailed in Algorithm 2. Importantly, in this algorithm, the sequence $\{u(k), u(k+1), ...u(k+\hat{d}-1)\}$ comprises control signals generated based on prior state inputs. These are utilized in the iterations to reach a more accurate state prediction.

10.2.3 Controller

In our proposed framework, we employ a controller designed using a delay-free plant model. Specifically, we opt for a Linear Quadratic Regulator (LQR) as the controller[KS72]. Unlike MPC, which necessitates online optimization during operation, the optimization in LQR is carried out offline based on the delay-free model

Algorithm 2: Smith Predictor in d steps

Input: $x(k)$, d, $\{u(k), u(k+1), ..., u(k+\hat{d}-1)\}$
$i \leftarrow 1$;
$e(k) \leftarrow x(k) - \hat{x}(k)$;
while $i \leq d$ **do**
  $\hat{x}(k+i) \leftarrow f(x(k+i-1), u(k+i-1)) + e(k)$;
  $x(k+i) \leftarrow \hat{x}(k+i)$;
  $i \leftarrow i+1$;
end
Output: $\hat{x}(k+\hat{d})$

Equation (10.2). Consequently, the control signal $u(k+\hat{d})$, scheduled to be actuated at time $k+\hat{d}$, is computed as shown in Equation (10.8). In this equation, K represents the LQR gain, pre-calculated based on offline optimization. The control signal is determined in the cloud-based control service by considering both the reference state x_{ref} and the predicted state $\hat{x}(k+\hat{d})$ for the plant at time $k+\hat{d}$.

$$u(k+\hat{d}) = K(x_{\text{ref}} - \hat{x}(k+\hat{d})) \tag{10.8}$$

It should be noted that while we have selected LQR due to its low online computational complexity, particularly advantageous when deployed in a cloud-based service, our framework is capable of accommodating other types of controllers as well.

10.3 Testbed Deployment

Now, we delve into the details of the testbed deployment for our punctual cloud framework. As depicted in Figure 10.4, the testbed consists of three key components: the plant under control, the network connecting the plant and the cloud services, and the cloud services constituting our punctual control framework. These are deployed as microservices within the Kubernetes cluster in our lab. The entire source code for this framework is available at GitLab[1].

10.3.1 The BnB plant

In our testbed, we consider an emulated BnB plant, which serves as the subject of control in the CCS. This emulation runs on an Ubuntu machine and is based on source

[1] `https://gitlab.com/Haorui/punctual-cloud`

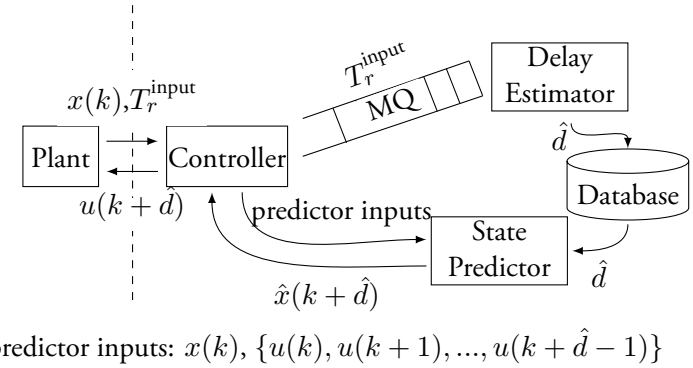


Figure 10.4: Punctual cloud deployment. The left part of the dashed line is the plant to be controlled, and the right part is the cloud services in our framework.

code provided by [ST]. The objective of the controller is to manipulate the angle of the beam in such a way as to maintain the ball at a predefined reference position. Given the fast dynamics of the BnB system, its control is considered time-sensitive [Ska+18]. It is worth noting that the plant could easily be replaced by different systems requiring other control strategies.

The BnB plant sends to the cloud services a state vector $x(k)$ with the period of T_s. The state vector consists of three elements: ball position, beam angle, and ball speed.

The BnB plant periodically transmits a state vector $x(k)$ to the cloud services at intervals of T_s. This vector is encompass three elements: the ball's position, the beam's angle, and the ball's speed, formalized as follows:

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} = \begin{bmatrix} \text{ball position} \\ \text{beam angle} \\ \text{ball speed} \end{bmatrix} \tag{10.9}$$

10.3.2 Network between the plant and the cloud services

We choose HTTP1 with a persistent connection to facilitate the communication between the plant and our service cluster. Since HTTP is the most commonly used and supported application protocol in cloud services[DJ21]. Accordingly, the plant's state vector $x(k)$ and the response delay T_r^{input} are packaged into an HTTP request and transmitted to the cloud services. The cloud then responds with a control signal as an HTTP response.

To emulate scenarios with extended network latency in the testbed, we employ Netem to artificially introduce additional delays on the Ethernet link connecting the client

machine and the cluster. This artificially added network delay, denoted as T_{add}, characterized by mean value μ and jitter δ. It augments the existing Ethernet connection latency, which comprises T_{up} and T_{down} as illustrated in Figure 10.1. Consequently, the actual response delays T_r exceed T_{add} as they also account for the time spent during cloud processing.

10.3.3 Punctual cloud services

The cloud application within our punctual cloud framework comprised of three interconnected services: the controller, delay estimator, and state predictor. These services operate synergistically in a microservice architecture, capitalizing on the diverse communication paradigms available in cloud-native environments.

Same as the previous implementation in Chapter 9, the services are all containerized and deployed on our lab Kubernetes cluster. Although the application architecture parallels the system put forth in Chapter 9, this implementation enhances inter-service communication reliability through the use of a Message Queue (MQ) within the microservice architecture.

In this layout, the **controller service** functions as the application's frontend, receiving the state periodically transmitted by the plant. It parses the incoming request to extract the embedded response delay T_r^{input} and publishes it to an MQ, to which the **delay estimator service** subscribes. The delay estimator then computes an estimated response delay $\hat{d}$ based on this most recent T_r^{input}, which serves as a gauge of the current network conditions between the plant and the cloud. Upon deriving $\hat{d}$, the delay estimator updates this value in a database accessible to the **state predictor** service.

After publishing the response delay to the MQ, the controller forwards the state $x(k)$ to the state predictor in the form of another HTTP request. Along with $x(k)$ is the previously generated control signal $u(k)$, calculated from a prior state prediction and already sent to the plant. When tasked with making a new prediction, the state predictor fetches the estimated delay $\hat{d}$ from the database. It then computes a predicted state $\hat{x}(k+\hat{d})$ based on received inputs and returns this value to the controller.

Finally, leveraging this newly acquired predicted state $\hat{x}(k+\hat{d})$, the controller formulates a new control signal $u(k+\hat{d})$. This signal is engineered to reach the plant before the expiry of the state $x(k+d)$ at time $k+\hat{d}+1$. If the control signal arrives ahead of time $k+\hat{d}$, the plant will buffer for actuation precisely at $k+\hat{d}$.

10.4 Evaluation

In this section, we provide the system parameters in our testbed and the performance metrics we used to evaluate our CCS under the punctual cloud framework.

We compare our punctual cloud framework with a MPC and a standalone LQR without punctual cloud framework. All three methods are deployed as cloud controllers in the same Kubernetes cluster, where MPC and LQR are deployed as a single service for each, and the punctual cloud framework is deployed in a microservice architecture. The experiments show that the punctual cloud framework can improve the system performance when discrete response delay $d \geqslant 1$, and that it does not add computational overheads with its microservice architecture.

10.4.1 Parameters of BnB plant

In the experiment, we employ a BnB plant as the target system for control. The BnB plant features a beam length of $1.1m$ and operates with a sampling time of $T_s = 50$ms. To model this system, we use a discretized, linear, and delay-free state-space representation, as expressed in Equation (10.2). The parameters for this model are specifically tuned to account for the beam length of 1.1m.

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0.05 & -0.008756 \\ 0 & 1 & -0.3502 \\ 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} -6.421e^{-5} \\ -0.003853 \\ 0.022 \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} D = 0 \end{aligned} \tag{10.10}$$

Giving the control signal $u(k)$ and state $x(k)$, the nonlinear physical model is formulated as:

$$\begin{aligned} x_2(k+1) &= 0.44 T_s u(k) + x_2(k) \\ M &= \frac{5}{7} g \sin(x_2(k+1)) \\ N &= \frac{5}{7} 0.44^2 u^2(k) \\ Q &= T_s x_3(k) + x_1(k) \\ x_3(k+1) &= \frac{NQ - M}{1 - BT_s^2} T_s + x_3(k) \\ x_1(k+1) &= T_s x_3(k+1) + x_1(k) \\ x(k+1) &= [x_1(k+1), x_2(k+1), x_3(k+1)]^T \end{aligned} \tag{10.11}$$

where $g = 9.80665$ is the gravitational constant.

For the evaluation, we conduct a step response experiment. The set point for the ball's position alternates between 0 and 0.2m to assess the system's responsiveness and stability under different network conditions. The control signal $u(k)$ aims to rotate the beam and maintain the ball at its set point. A key stability criterion is that if the ball's position $x_1(k)$ reaches or exceeds 0.55m, the system becomes unstable, and the ball falls off the beam.

10.4.2 Network parameters

In our experiments, we focus on evaluating the system under a variety of network conditions, specifically by manipulating the additional network delay parameter T_{add} characterized by its mean value μ and jitter δ.

Additional network delay parameters

- **Mean delay** (μ): We vary μ from 0 to 200ms, up to 4 times the BnB plant's sampling time T_s.
- **Jitter** (δ): The jitter values range from 0 to 25ms, equal to $0.5T_s$

The controller response delay T_r will exceed the added network delay T_{add} due to the additional execution time required by the cloud-based controller.

Network scenarios

Two main network scenarios are considered in our evaluation:

- **Changing delay scenario**: In this scenario, the mean delay value μ is dynamically altered while the controller is operational. Concurrently, the set point for the ball's position alternates between two distinct positions. The objective is to evaluate the delay estimator's adaptability and its ability to furnish the state predictor with accurate, real-time data.
- **Fixed delay scenarios**: In this scenario, T_{add} is modelled using a Pareto distribution based on a specific pair of (μ, σ) values. We subdivide this into three categories:

1. **Increasing mean, no jitter**: T_{add} assumes μ values from the set $\{0, T_s, 2T_s, 3T_s, 4T_s\}$ with $\delta = 0$.
2. **Increasing jitter, fixed mean**: T_{add} adopts δ values from the set $\{0, 0.1T_s, 0.2T_s, 0.3T_s, 0.4T_s, 0.5T_s\}$ while keeping μ fixed at $3T_s$.
3. **Increasing mean, fixed jitter**: T_{add} assumes μ value from the set $\{0, T_s, 2T_s, 3T_s, 4T_s\}$ and a fixed jitter value $\delta = 0.5T_s$.

10.4.3 Punctual cloud parameters

In this section, we detail the parameters utilized for both the delay estimator and the controller. It is important to note that the state predictor operates exclusively based on the plant's nonlinear model and does not require additional parameters beyond the inputs from the controller and delay estimator.

The weight parameter employed in our delay estimator within the testbed is $\omega = 0.98$. The control signal $u(k)$ actuates on the angular velocity of the beam and is determined using the LQR gain K, given a target ball position x_{ref} on the beam. Mathematically, this is expressed as follows:

$$\begin{aligned} u(k) &= K\left(\begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} - \begin{bmatrix} x_{\text{ref}}(k) \\ 0 \\ 0 \end{bmatrix}\right) \\ K &= \begin{bmatrix} -31.0027, -13.9077, 20.4937 \end{bmatrix} \end{aligned} \tag{10.12}$$

10.4.4 Performance metrics

For each network scenario described in Section 10.4.2, we assess the system performances using the punctual cloud framework, as well as MPC and LQR control methods. Our evaluation focuses on three key perspectives:

1. **Control Performance**: The is evaluated using the Integral Absolute Error (IAE). A lower IAE value indicates better control performance, characterized by smaller deviations from the set point. The IAE is computed over an experimental time T as follows:

$$\text{IAE} = \sum_{k=0}^{T} \left| x_1(k) - x_{\text{ref}}(k) \right|, \tag{10.13}$$

   Where $x_1(k)$ represents the ball's position state, and $x_{\text{ref}}(k)$ is its position setpoint.

Table 10.1: Parameters in the experiments.

T_{add}	Network delay added by Netem
u	Mean value of T_{add}
δ	Jitter value of T_{add}
T_s	Sampling time of the plant
T_{exe}	Execution time of the services in the cloud
$T_r, \hat{T}_r$	Response delay of control signals and its estimation
$d, \hat{d}$	Response delay in discrete time and its estimation

2. **Execution Time**: This measures the computational overhead incurred by each method in generating a control signal. As we implement a persistent HTTP connection in our testbed, the execution time T_{exe} can be calculated according to Equation (10.14), which involves the measured response delay T_r for each HTTP request-response pair.

$$T_{\text{exe}} = T_r - T_{\text{add}} \tag{10.14}$$

3. **Punctuality**: We define punctuality as the ratio of control signals that arrive timely during an experiment. A control signal $u(k)$ is considered "on time" if it arrives and is actuated at the plant before the associated state $x(k)$ expires. Higher punctuality values indicate more timely actuation of control signals.

The parameters and their corresponding notations used in our experiments are detailed in Table 10.1.

10.5 Evaluation Results

In this section, we present the performance results under various scenarios. The evaluation includes comparisons between an MPC, a standalone LQR, and our proposed punctual cloud framework, which is abbreviated as 'PunC' in the following presented figures.

10.5.1 Changing delay scenario

In the evaluation scenario with changing network delay, we conduct tests where the mean value, μ, of the added network delay T_{add} varies within the range of 0 and $4T_s$. This variation is illustrated in the lower portion of Figure 10.5. As the results indicate, the standalone LQR controller maintains stability with delays up to $2T_s$, while the MPC method can handle delays up to $3T_s$, corresponding to a response delay $d = 3$.

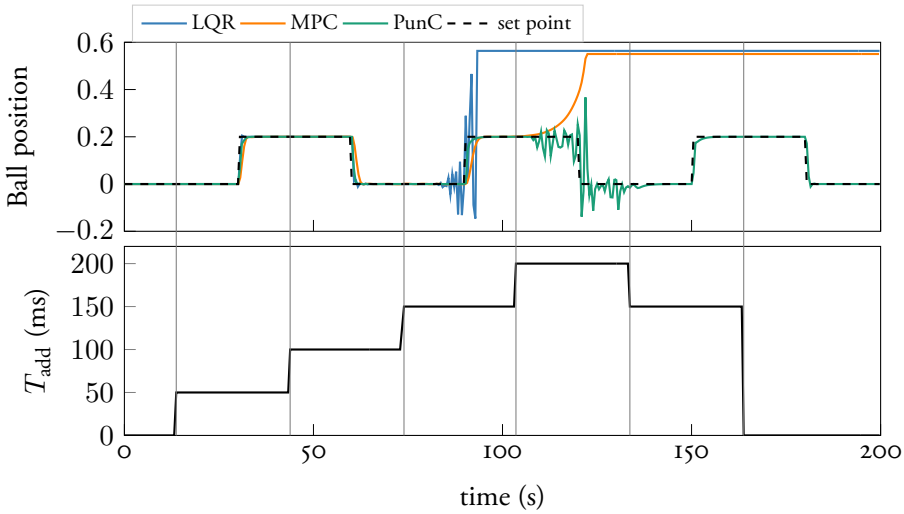


Figure 10.5: The top figure depicts the control step response performance as it varies with changing added network delay, T_{add}, which follows the pattern illustrated in the bottom figure.

However, for $T_{\text{add}} > 3T_s$, or equivalently $d > 3$, both the MPC and LQR approaches fail to maintain the ball's position at its designated set point.

In contrast, our Punctual Cloud framework adapts to the network conditions by generating control signals $u(k)$ based on the predicted state $\hat{x}(k)$. This approach ensures the timely arrival of the control signals to stabilize the actual state of the plant, $x(k)$, under the assumption that the network is lossless.

10.5.2 Fixed delay scenario

For this scenario, we run our evaluation when the mean and jitter of T_{add} are fixed in Netem configuration for individual experiment. We analyse our system performance under the following three sub-scenarios:

Analysis on increasing meaning and no jitter

In Figure 10.6, we show the step response under various levels of added delay T_{add} when the jitter is 0. As we can see, the standalone LQR fails to stabilize the plant when $T_{\text{add}} = 3T_s$, causing the ball falls off the beam. Similarly, MPC loses control over the ball's position when $T_{\text{add}} = 4T_s$. In contrast, our proposed punctual cloud framework effectively maintains the ball near its reference position up to $4T_s$, although with some oscillation. This shows our framework's capability for achieving a four-step-ahead

Figure 10.6: The first row of the figure is the step response performance when (a) $T_{\text{add}} = 2T_s$ (b) $T_{\text{add}} = 3T_s$ and (c) $T_{\text{add}} = 4T_s$; the second row are the punctual cloud state prediction errors presented in percentage.

prediction in discrete time.

The corresponding error plot Figure 10.6 reveals the performance limitation of the state predictor, especially when T_{add} reaches $4T_s$. At this point, prediction errors become significant, leading to control performance degradation and oscillation. This suggests that a more accurate state predictor could potentially improve control performance even at delays exceeding $4T_s$.

Figure 10.7 summaries the performance for each control strategies on execution time, IAE and punctuality, with increasing T_{add} from 0 to $4T_s$. Among the methods, LQR has the shortest execution time due to its minimal computational complexity, equivalent to a P-controller. MPC, although also deployed as a standalone service, requires more computational time because of its online optimization process. In contrast, our punctual cloud framework, although built upon a more complex microservice architecture, manages to maintain relatively low response delays thanks to its lightweight services.

As T_{add} increases, of all methods exhibit a rising IAE, implying control performance degradation. Once T_{add} surpasses $3T_s$, both MPC and LQR returns exceedingly high IAE values, exceeding 1000, indicating system instability. Remarkably, our punctual cloud approach maintains a lower error level under the same conditions.

Finally, the punctuality plot in Figure 10.7 shows a marked improvement in our framework's punctuality performance, attributed to the incorporation of delay estimation and state prediction. Our method ensures that control signals $u(k)$ are mostly applied in a timely manner, substantially enhancing the system's overall control performance.

Analysis on increasing δ when $\mu = 3T_s$

In these experiments, we focus exclusively on our proposed punctual cloud framework, as the standalone LQR fails to stabilize the system when T_{add} mean value μ is $3T_s$, even with zero jitter. The evaluation varies the jitter δ from 0 to $0.5T_s$, while keeping $\mu = 3T_s$.

The box plots in Figure 10.8 presents the response time T_r measurements, demonstrating the system's behaviour under increased jitter. Although the median value of the response delays remains constant, we observe larger variances and a significantly higher 95th percentile value as jitter increases. These trends present a challenge for the EMA algorithm employed for delay estimation in maintaining punctuality.

The IAE plot in the same figure reveals a direct correlation between increased jitter and

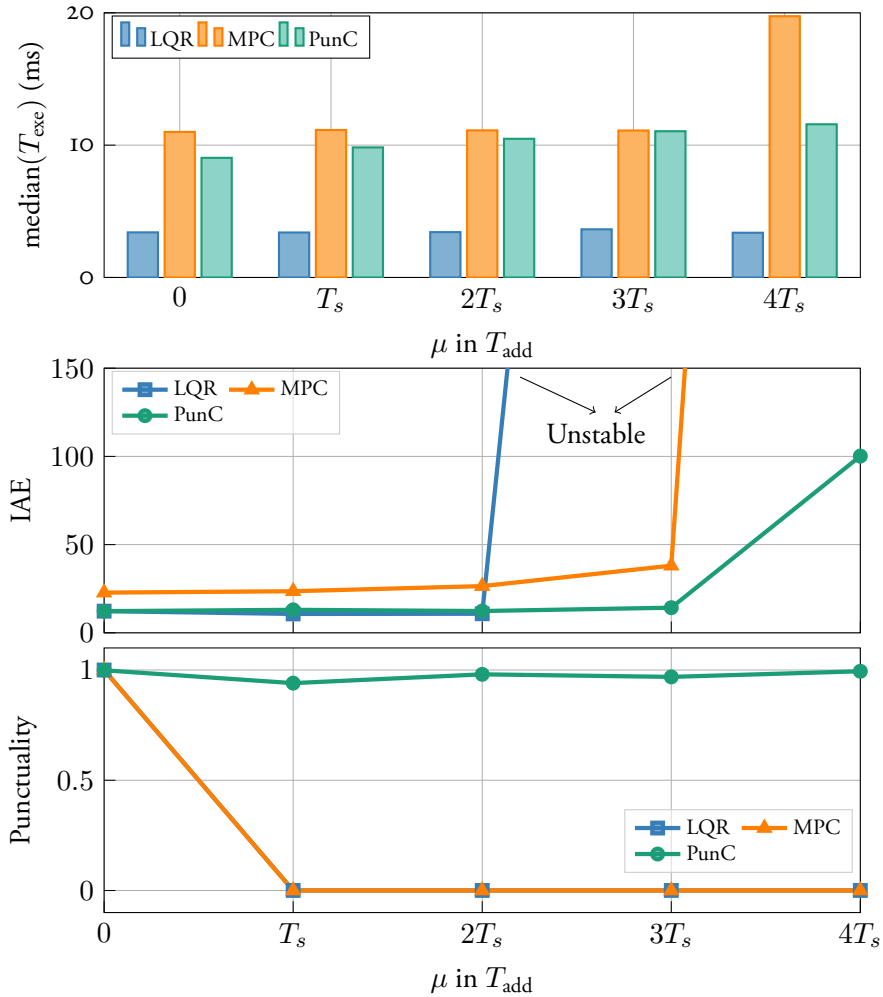


Figure 10.7: Performances of three methods when delay increases and no jitter in T_{add}. From top to bottom, each figure presents the execution time, IAE, and punctuality with each method in the evaluation.

higher IAE values, even though the system remains stable under the maximum jitter conditions tested. This degradation in control performance is further explained by the punctuality plot. The plot clearly shows that a decline in punctuality corresponds to an increase in IAE values.

This decline in punctuality affects the control performance in two key ways:

1. Control signals may actuate on a plant state that has significantly diverged from the state on which the signal was originally calculated for.

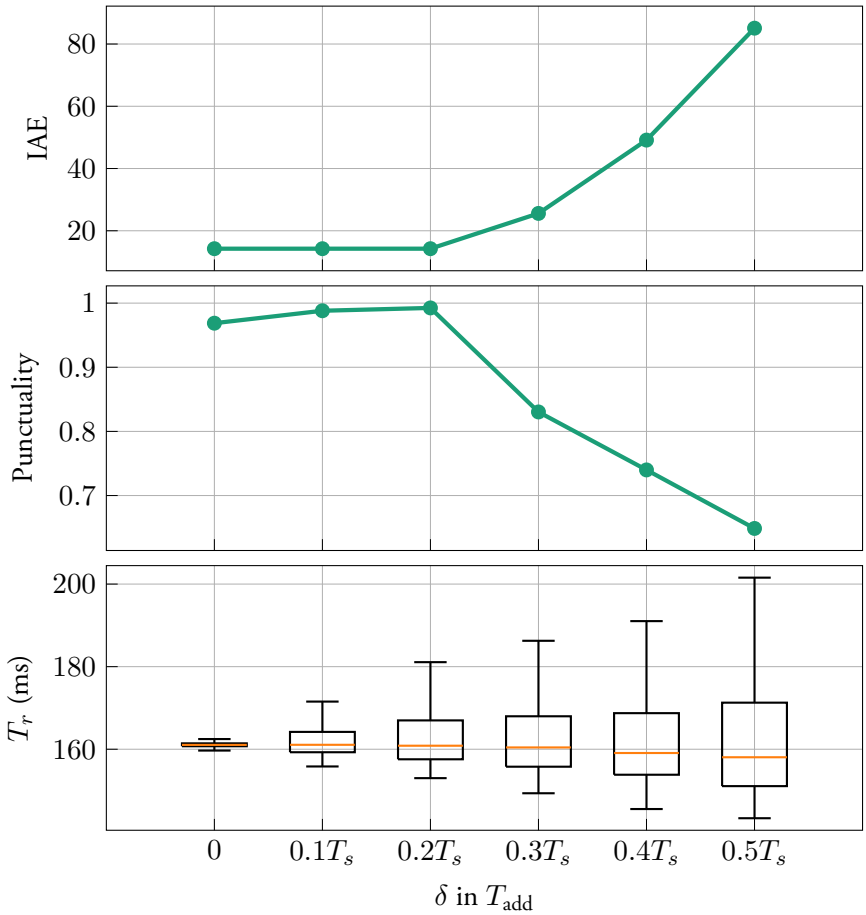


Figure 10.8: The first two sub-figures are IAE and punctuality performances of punctual cloud when jitter increases and added network delay mean fixed to $3T_s$. The bottom figure is the box plots of response delays in each experiment.

2. Reduced punctuality also compromises the accuracy of the state predictor. The predictor struggles to determine which control signals were actually used in the duration between states $x(k)$ and $x(k+d)$, leading to less accurate predictions.

Analysis on increasing μ with $\delta = 0.5T_s$

Figure 10.9 compares the performance of all three methods across increasing T_{add} mean value μ while maintaining jitter δ at $0.5T_s$ in Netem. The added delay T_{add} are subject to a Pareto distribution.

Similarly to the previous case, in this evaluation, both MPC and LQR fail when faced

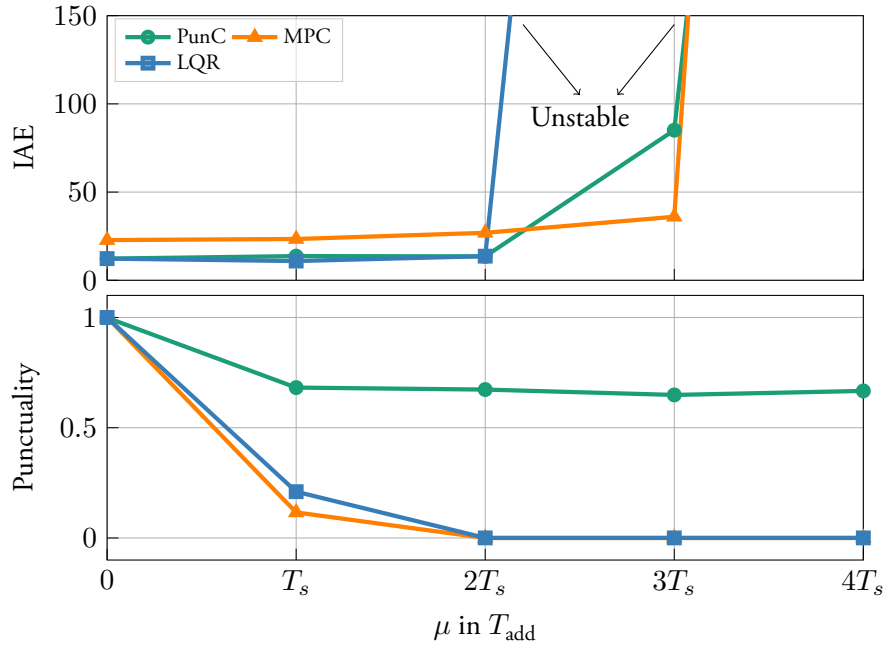


Figure 10.9: IAE and punctuality performances of three methods with increasing μ and δ fixed to $0.5T_s$

with an added network delay that results in a discrete-time response delay of more than 3 steps. More notably, even the punctual cloud framework fails to stabilize the system when the added delay mean μ is $4T_s$.

The root cause of this instability is illustrated in Figure 10.10, where it becomes evident that higher jitter levels significantly impair the accuracy of delay estimation using EMA. This inaccuracy, in turn, diminishes both the punctuality of control signals and the precision of state predictions. Furthermore, as shown in Figure 10.6(c), when $\mu = 4T_s$, the ball begins to oscillate around its reference position. This oscillation is exacerbated by the larger jitter, introducing greater uncertainties and inaccuracies in state prediction, ultimately destabilizing the control system.

10.6 Conclusion on Punctual Cloud for CCSs

This chapter presents an adaptation of the punctual cloud framework tailored for Cloud Control Systems (CCSs). Typically, CCSs operate as time-delayed control systems, where a control signal $u(k)$ generated in the cloud is not applied instantaneously to the plant state $x(k)$, but rather at $x(k + d)$, owing to the presence of a response delay d.

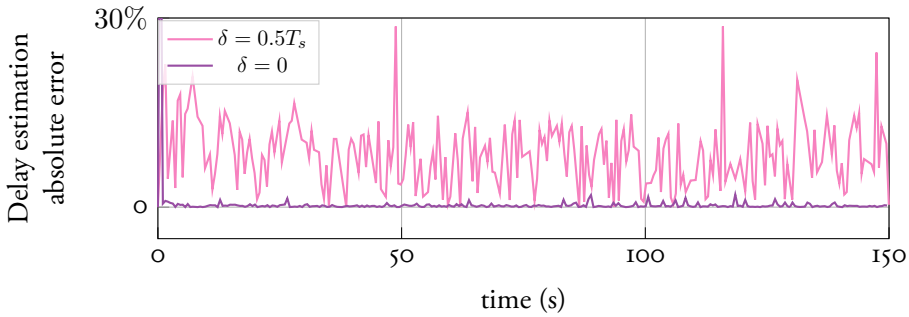


Figure 10.10: The Absolute delay estimation error in punctual cloud when jitter $\delta = 0$ and $\delta = 0.5T_s$

Uniquely, our punctual cloud framework offers the advantage of utilizing a straightforward controller based on a delay-free system model, eliminating the need to modify the controller to accommodate system latencies. We address the delay issue by integrating two supplementary services: a delay estimator and a state predictor. These services collaboratively estimate the control signal's arrival time at the plant and calculate the corresponding control signal based on the predicted plant state.

Our evaluations employed a BnB plant emulator with a sampling interval of 50ms. We deployed our framework in microservice architecture in our lab Kubernetes cluster and emulate various network delays using Netem. The delay estimator employed an EMA algorithm, while a Smith Predictor was used for state prediction. Importantly, these services can be readily swapped out for other algorithms without requiring adjustments to the remaining system components.

Comparative evaluations with MPC and standalone LQR demonstrated the advantages of our method. Specifically, our punctual cloud framework could tolerate added network delays up to $4T_s$ for time-sensitive applications, requiring predictions up to 4 steps ahead in discrete time. In contrast, both MPC and LQR were limited to tolerating delays up $3T_s$. Furthermore, our method exhibited resilience to network jitter up to 25ms under a Pareto distribution, provided the network delay remained below $4T_s$.

Summary

In the final part, we introduced a framework named as "Punctual Cloud", designed to mitigate the challenges posed by cloud-induced delays in time-critical cloud integrated systems.

Our framework was validated through two distinct use-cases: radio resource allocation in Cloud RAN and Cloud Control System (CCS). The punctual cloud framework aims to compensate the performance degradation brought about by long response delays of real-time cloud applications. These applications are particularly sensitive to latency; however, the unpredictable nature of network conditions and cloud environments often fails to meet the stringent requirements of such demanding applications. Against this backdrop, the punctual cloud framework emerges as a robust solution for mitigating the impacts of long and uncertain delays in real-time and mission-critical cloud integrated systems.

It should be highlighted that, our framework may not be necessary in environments where the network and cloud infrastructure can consistently guarantee low latency and fulfil the QoS requirements. In such cases, introducing additional services and elongated data path to remedy the other performance properties of the application may be superfluous, as the performance would already be near-ideal–comparable to scenarios where the client and computing tasks are co-located.

In summary, the punctual cloud framework specifically targets cloud applications suffering from unacceptable levels of long and uncertain response delays. Our evaluations demonstrate marked performance improvements under these challenging conditions across both use-cases. Furthermore, the framework's modularity allows for the easy incorporation of more accurate estimation and prediction algorithms, providing avenues for ongoing research and system optimization

Part V

Conclusion

Chapter 11

Conclusions on my PhD study

As the title might hint, I consider this thesis less a formal document and more a personal journal, capturing my initial exploration into the intricate realm of cloud integrated systems and my attempts to tame these complex beasts in the wild.

This odyssey has been anything but linear, with many detours and branching paths. Yet, it has been an interesting exploration in an era defined not just by clouds and networks, but various interconnected systems. I've summarized my PhD experience into three core themes: Cloud RAN, Cloud Control System, and the Punctual Cloud.

The early stages of my PhD were spent in the domain of RAN and Cloud RAN, particularly within the context of indoor industrial settings. Back then, our access to both cloud and network infrastructure were limited. We had to rely heavily on simulations and assumptions. During this time, we delved into the potential of deploying massive MIMO RAN and Cloud RAN in an industrial environment, focusing mainly on the problem of uplink pilot allocation. While conclusions could vary with different system setups, the key takeaway was the feasibility of our proposed architecture and the delivery of a simulation model that can be repurposed for other similar scenarios.

As my interests evolved, so did my enthusiasm in research topics—shifting toward the fascinating realm of cloud integrated systems, more specifically, Cloud Control System. At this stage, I gained more access to real-world network and cloud resources, and I was able to engage with the complexities of deploying CCSs in the wild. This phase endowed me with more insights about the systems: from understanding the nuances of "controllers over the cloud" to familiarizing myself with mobile networks, diverse network protocols, and an array of cloud services and architectures. I also gained more understanding about 5G, which is quite common nowadays, but was viewed as a magic cure to all time-critical networked systems due to its "1ms latency

myth" a few years ago. Though 5G and its edge fell short of this ideal, it still offered a great opportunity for innovation, demanding interdisciplinary research to improve cloud integrated system performance and adaptability. The pivotal contribution of my study at this stage was a testbed that served to demonstrate the practical viability of deploying CCSs using cloud-native services, particularly over 5G networks. Based on the empirical evidence gathered, we assessed the system's performance from various angles within the network, particularly at the radio and transport layers, in a quest to identify an optimal network configuration that could coexist harmoniously within a cloud-native ecosystem while providing robust system control.

This investigative trajectory naturally culminated in the conceptualization of what I call the "Punctual Cloud" framework. While perhaps not the most intricate cloud application developed during my PhD study, it encapsulates my overarching vision for the modular deployment of control systems in a cloud-native manner. While it is conceivable that the challenges observed in CCS could be addressed via more advanced control algorithms or machine learning techniques, I believe that the "Punctual Cloud" offers an elegantly simple and modular solution for cloud integrated system deployment. The framework not only counteracts network latencies inherent in real-world scenarios but also streamlines the deployment and ongoing maintenance of cloud services for the systems, thereby facilitating effortless modular development and upgrades.

As I close the chapter on my PhD journey, the exploration of modular cloud integrated systems is only just getting started. The quest for optimizing networked, cloud-based, time-critical systems is far from complete, as they may never reach the performance of their counterparts where all modules are colocated. Furthermore, fully integrating cloud and mobile networks into traditional industrial settings still remains enormous efforts, extending well beyond just the plant devices and their controllers.

I hope you've enjoyed this literary excursion. Thank you for reading.

Bibliography

[Smi57] O. Smith. "Closer Control of Loops with Dead Time". In: 1957.

[KS72] Huibert Kwakernaak and Raphael Sivan. "Linear Optimal Control Systems". In: 1972.

[Ric+78] J. Richalet, A. Rault, J.L. Testud, and J. Papon. "Model predictive heuristic control: Applications to industrial processes". In: *Automatica* 14.5 (1978), pp. 413–428. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(78)90001-8.

[Gar99] Simson Garfinkel. *Architects of the information society: 35 years of the Laboratory for Computer Science at MIT*. Cambridge, MA: MIT Press, 1999. ISBN: 0585054886.

[Nie+99] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. DOI: 10.17487/RFC2616.

[Por+06] Scott Poretsky, Shobha Erramilli, Jerry Perser, and Sumit Khurana. *Terminology for Benchmarking Network-layer Traffic Control Mechanisms*. RFC 4689. Oct. 2006. DOI: 10.17487/RFC4689.

[Zho06] Qing Chang Zhong. *Robust Control of Time-delay Systems*. Springer London, 2006. ISBN: 1-84628-264-0. DOI: 10.1007/1-84628-265-9.

[For07] Behrouz A. Forouzan. *Data Communications and Networking*. Data Communications and Networking. McGraw-Hill Higher Education, 2007. ISBN: 9780072967753.

[07] "Model Predictive Control of Dead-time Processes". In: *Control of Dead-time Processes*. London: Springer London, 2007, pp. 271–308. ISBN: 978-1-84628-829-6. DOI: 10.1007/978-1-84628-829-6_9.

[Dre08] Ulrich Drepper.
"The Cost of Virtualization: Software Developers Need to Be Aware of the Compromises They Face When Using Virtualization Technology."
In: *Queue* 6.1 (Jan. 2008), pp. 28–35. ISSN: 1542-7730.
DOI: 10.1145/1348583.1348591.

[BI09] Chandra S. Bontu and Ed Illidge.
"DRX mechanism for power saving in LTE".
In: *IEEE Communications Magazine* (June 2009). ISSN: 0163-6804.
DOI: 10.1109/MCOM.2009.5116800.

[Tsa+10] Wei-Tek Tsai, Qihong Shao, Xin Sun, and Jay Elston.
"Real-Time Service-Oriented Cloud Computing".
In: *2010 6th World Congress on Services*. IEEE, July 2010, pp. 473–478.
ISBN: 978-1-4244-8199-6. DOI: 10.1109/SERVICES.2010.127.

[WN10] Guohui Wang and T. S. Eugene Ng. "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center". In:
IEEE, Mar. 2010, pp. 1–9. ISBN: 978-1-4244-5836-3.
DOI: 10.1109/INFCOM.2010.5461931.

[Chi11] China Mobile. *C-RAN: The Road towards Green RAN*.
China Mobile White Paper. Last accessed Nov. 2020. 2011.
URL: https://www.semanticscholar.org/paper/C-ran-the-Road-towards-Green-Ran/eaa3ca62c9d5653e4f2318aed9ddb8992a505d3c.

[MG11] Peter Mell and Timothy Grance.
The NIST Definition of Cloud Computing. 2011.
DOI: SpecialPublication800-145.

[Gua+12] L. Guangjie, Z. Senjie, Y. Xuebin, L. Fanglan, N. Tin-fook, Z. Sunny, and K. Chen.
"Architecture of GPP based, scalable, large-scale C-RAN BBU pool".
In: *2012 IEEE Globecom Workshops*. Dec. 2012, pp. 267–272.
DOI: 10.1109/GLOCOMW.2012.6477581.

[Xu12] Xun Xu. "From cloud computing to cloud manufacturing". In: *Robotics and Computer-Integrated Manufacturing* 28.1 (Feb. 2012), pp. 75–86.
ISSN: 07365845. DOI: 10.1016/j.rcim.2011.07.002.

[GCL14] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu.
"Challenges in real-time virtualization and predictable cloud computing".
In: *Journal of Systems Architecture* 60 (9 Oct. 2014), pp. 726–740.
ISSN: 13837621. DOI: 10.1016/j.sysarc.2014.07.004.

[GLK14] Aditya Gudipati, Li Erran Li, and Sachin Katti.
"RadioVisor: A Slicing Plane for Radio Access Networks".
In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14.
Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 237–238. ISBN: 9781450329897.
DOI: 10.1145/2620728.2620782.

[Vie+14] J. Vieira, S. Malkowsky, K. Nieman, Z. Miers, N. Kundargi, L. Liu, I. Wong, V. Öwall, O. Edfors, and F. Tufvesson.
"A flexible 100-antenna testbed for Massive MIMO".
In: *2014 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2014, pp. 287–293. DOI: 10.1109/GLOCOMW.2014.7063446.

[BPT15] Mike Belshe, Roberto Peon, and Martin Thomson.
Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. May 2015.
DOI: 10.17487/RFC7540.

[Ese+15] Hasan Esen, Masakazu Adachi, Daniele Bernardini, Alberto Bemporad, Dominik Rost, and Jens Knodel. "Control as a service (CaaS)".
In: *Proceedings of the Second International Workshop on the Swarm at the Edge of the Cloud*. New York, USA: ACM, Apr. 2015, pp. 13–18.
ISBN: 9781450335959. DOI: 10.1145/2756755.2756758.

[Hu+15] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young.
"Mobile Edge Computing - A key technology towards 5G".
In: *ETSI White Paper No. 11*. Last Accessed: August 14, 2023. 2015.
ISBN: 979-10-92620-08-5.
URL: https://www.etsi.org/images/files/ETSIWhitePapers/etsi%5C%5Fwp11%5C%5Fmec%5C%5Fa%5C%5Fkey%5C%5Ftechnology%5C%5Ftowards%5C%5F5g.pdf.

[Buy+16] Rajkumar Buyya, Amir Vahid Dastjerdi, Feng Xia, Laurence T. Yang, Lizhe Wang, and Alexey Vinel.
Internet of Things: Principles and Paradigms. 1st.
San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
ISBN: 978-0-12-805395-9.

[Geo+16] Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Maria Fazia, Massimo Villari, and Rajiv Ranjan. "Internet of Things and Edge Cloud Computing Roadmap for Manufacturing".
In: *IEEE Cloud Computing* 3.4 (July 2016), pp. 66–73.
ISSN: 2325-6095. DOI: 10.1109/MCC.2016.91.

[Ha+16] Son-Hai Ha, Daniele Venzano, Patrick Brown, and Pietro Michiardi. "On the impact of virtualization on the I/O performance of analytic workloads". In: *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*. IEEE, May 2016, pp. 31–38. ISBN: 978-1-4673-8894-8. DOI: 10.1109/CloudTech.2016.7847722.

[Mar+16] Thomas L. Marzetta, Erik G. Larsson, Hong Yang, and Hien Quoc Ngo. *Fundamentals of Massive MIMO*. Cambridge University Press, Nov. 2016. ISBN: 9781107175570. DOI: 10.1017/CBO9781316799895.

[Mat+16] Michał Maternia, Salah Eddine El Ayoubi, Mikael Fallgren, Panagiotis Spapis, Yinan Qi, David Martín-Sacristán, Óscar Carrasco, Maria Fresia, Miquel Payaró, Martin Schubert, Jean Sébastien Bedo, and Vivek Kulkarni. *5G PPP use cases and performance evaluation models*. Tech. rep. Accessed: Oct. 23, 2019. 5Gppp, Apr. 2016. URL: http://www.5g-ppp.eu/.

[PL16] Jakub Pizoń and Jerzy Lipski. "Perspectives for Fog Computing in Manufacturing". In: *Applied Computer Science* 12.3 (2016), pp. 37–46. URL: http://acs.pollub.pl/pdf/v12n3/4.pdf.

[WZM16] Kaiwei Wang, Wuyang Zhou, and Shiwen Mao. "Energy Efficient Joint Resource Scheduling for Delay-Aware Traffic in Cloud-RAN". In: *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec. 2016. ISBN: 978-1-5090-1328-9. DOI: 10.1109/GLOCOM.2016.7841793.

[1417] 3GPP-Release 14. *Study on new radio access technology: Radio access architecture and interfaces*. Technical Report 3GPP TR 38.801 V14.0.0 (2017-03). 3GPP, 2017.

[Aij17] Adnan Aijaz. "A Radio Resource Slicing Framework for 5G Networks With Haptic Communications". In: *IEEE Systems Journal* 12.3 (2017), pp. 2285–2296. ISSN: 1932-8184. DOI: 10.1109/jsyst.2017.2647970.

[Ass+17] Philippos Assimakopulos, Gurtej S. Birring, M. Kenan Al-Hares, and Nathan J. Gomes. "Ethernet-based fronthauling for cloud-radio access networks". In: *2017 19th International Conference on Transparent Optical Networks (ICTON)*. 2017, pp. 1–4. DOI: 10.1109/ICTON.2017.8025034.

[FDA17] Lilatul Ferdouse, Olivia Das, and Alagan Anpalagan. "Auction Based Distributed Resource Allocation for Delay Aware OFDM Based Cloud-RAN System". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Vol. 2018-Janua. IEEE, Dec. 2017. ISBN: 978-1-5090-5019-2. DOI: 10.1109/GLOCOM.2017.8254627.

[GLA17] Mikael Gidlund, Tomas Lennvall, and Johan Akerberg. "Will 5G become yet another wireless technology for industrial automation?" In: *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, Mar. 2017, pp. 1319–1324. ISBN: 978-1-5090-5320-9. DOI: 10.1109/ICIT.2017.7915554.

[GSM17] GSMA. *An Introduction to Network Slicing*. Tech. rep. Accessed: Oct. 23, 2019. GSM Association, 2017. URL: https://www.gsma.com/futurenetworks/resources/an-introduction-to-network-slicing-2/.

[KN17] Adlen Ksentini and Navid Nikaein. "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction". In: *IEEE Communications Magazine* 55.6 (2017), pp. 102–108. ISSN: 01636804. DOI: 10.1109/MCOM.2017.1601119.

[Mal+17] Steffen Malkowsky, Joao Vieira, Liang Liu, Paul Harris, Karl Nieman, Nikhil Kundargi, Ian Wong, Fredrik Tufvesson, Viktor Öwall, and Ove Edfors. "The World's First Real-Time Testbed for Massive MIMO: Design, Implementation, and Validation". English. In: *IEEE Access* (2017), pp. 9073–9088. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2705561.

[MWY17] Haibo Mei, Kezhi Wang, and Kun Yang. "Multi-Layer Cloud-RAN With Cooperative Resource Allocations for Low-Latency Computing and Communication Services". In: *IEEE Access* 5 (2017), pp. 19023–19032. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2752279.

[Mou+17] Ghizlane Mountaser, Maria Lema Rosas, Toktam Mahmoodi, and Mischa Dohler. "On the feasibility of MAC and PHY split in cloud RAN". In: *IEEE Wireless Communications and Networking Conference, WCNC* (2017), pp. 1–6. ISSN: 15253511. DOI: 10.1109/WCNC.2017.7925770.

[Ord+17] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". In: *IEEE Communications Magazine* 55.5 (May 2017), pp. 80–87. ISSN: 1558-1896. DOI: 10.1109/MCOM.2017.1600935.

[Pan+17] Ai-Chun Pang, Wei-Ho Chung, Te-Chuan Chiu, and Junshan Zhang. "Latency-Driven Cooperative Task Computing in Multi-user Fog-Radio Access Networks". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 615–624. DOI: 10.1109/ICDCS.2017.83.

[Tey+17] Oumer Teyeb, Gustav Wikström, Magnus Stattin, Thomas Cheng, Sebastian Faxér, and Hieu Do. *Evolving LTE to fit the 5G future*. Tech. rep. Last Accessed: August 4, 2023. Ericsson, 2017. URL: https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/evolving-lte-to-fit-the-5g-future#:~:text=The%20process%20of%20making%20LTE,and%20user%20planes%20(UPs)..

[ZSG17] Giuliana Zennaro, Aleksandra Stojanovic, and Marina Giordanino. *Report on vertical requirements and use cases*. Tech. rep. 761536. Accessed: Mar, 4, 2020. 5G-TRANSFORMER Project, 2017. URL: http://5g-transformer.eu/index.php/deliverables/.

[Årz+18] Karl-Erik Årzén, Per Skarin, William Tärneberg, and Maria Kihl. "Control over the Edge Cloud - An MPC Example". English. In: *1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems*. Nashville, Tennessee, United States, Dec. 2018.

[Bek+18] C. Bektas, S. Monhof, F. Kurtz, and C. Wietfeld. "Towards 5G: An Empirical Evaluation of Software-Defined End-to-End Network Slicing". In: *2018 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2018, pp. 1–6. DOI: 10.1109/GLOCOMW.2018.8644145.

[BSK18] Christer Boberg, Malgorzata Svensson, and Benedek Kovács. *Distributed cloud –a key enabler of automotive and industry 4.0 use cases*. Tech. rep. Last Accessed: November 1, 2023. Ericsson, 2018.

[Gom+18] Nathan J. Gomes, Philippe Sehier, Howard Thomas, Philippe Chanclou, Bomin Li, Daniel Munch, Philippos Assimakopoulos, Sudhir Dixit, and Volker Jungnickel. "Boosting 5G Through Ethernet: How Evolved Fronthaul Can Take Next-Generation Mobile to the Next Level".

In: *IEEE Vehicular Technology Magazine* 13.1 (2018), pp. 74–84. DOI: 10.1109/MVT.2017.2782358.

[Hao+18] Yixue Hao, Yinging Jiang, M Shamim Hossain, Mohammed F Alhamid, and Syed Umar. "Learning for Smart Edge : Cognitive Learning-Based Computation Offloading". In: (2018).

[HMH18] Tobias Hosfeld, Florian Metzger, and Poul E. Heegaard. "Traffic modeling for aggregated periodic IoT data". In: *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, Feb. 2018. ISBN: 978-1-5386-3458-5. DOI: 10.1109/ICIN.2018.8401624.

[Ior+18] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Ned Goren, and Charif Mahmoudi. *Fog computing conceptual model*. Mar. 2018. DOI: 10.6028/NIST.SP.500-325.

[MMB18] Charif Mahmoudi, Fabrice Mourlin, and Abdella Battou. "Formal definition of edge computing: An emphasis on mobile cloud and IoT composition". In: IEEE, Apr. 2018, pp. 34–42. ISBN: 978-1-5386-5896-3. DOI: 10.1109/FMEC.2018.8364042.

[MRB18] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. "Latency-aware application module management for fog computing environments". In: *ACM Transactions on Internet Technology* 19.1 (2018). ISSN: 15576051. DOI: 10.1145/3186592.

[ODo+18] Peter O'Donovan, Colm Gallagher, Ken Bruton, and Dominic T.J. O'Sullivan. "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications". In: *Manufacturing Letters* 15 (Jan. 2018), pp. 139–142. ISSN: 22138463. DOI: 10.1016/j.mfglet.2018.01.005.

[Ska+18] Per Skarin, William Tärneberg, Karl-Erik Årzén, and Maria Kihl. "Towards Mission-Critical Control at the Edge and Over 5G". English. In: *2018 IEEE International Conference on Edge Computing (EDGE)*. United States: IEEE Computer Society, July 2018, pp. 50–57. ISBN: 978-1-5386-7238-9. DOI: 10.1109/EDGE.2018.00014.

[Cis19] Cisco. *Leading Tools Manufacturer Transforms Operations with IoT*. Last accessed Nov. 2020. 2019. URL: http://www.cisco.com/c/dam/en%5C%5Fus/solutions/industries/docs/manufacturing/c36-732293-00-stanley-cs.pdf.

[Diz+19] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin.
"A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration".
In: *ACM Comput. Surv.* 51.6 (6 Jan. 2019). ISSN: 0360-0300.
DOI: 10.1145/3292674.

[Gro19] ATIS IoT Categorization Focus Group. *IoT Categorization : Exploring the Need for Standardizing Additional Network Slices*.
Tech. rep. ATIS-I-0000075. Last accessed Nov. 2020. Alliance for Telecommunications Industry Solutions (ATIS), Sept. 2019.
URL: https://access.atis.org/apps/group%5C%5Fpublic/download.php/51129/ATIS-I-0000075.pdf.

[Seu+19] Michael Seufert, Raimund Schatz, Nikolas Wehner, and Pedro Casas.
"QUICker or not? -an Empirical Analysis of QUIC vs TCP for Video Streaming QoE Provisioning". In: *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2019.
ISBN: 978-1-5386-8336-1. DOI: 10.1109/ICIN.2019.8685913.

[Ska+19] Per Skarin, Karl-Erik Årzén, Johan Eker, and Maria Kihl.
"Cloud-Assisted Model Predictive Control". English.
In: *2019 IEEE International Conference on Edge Computing*.
United States: IEEE - Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 110–112. DOI: 10.1109/EDGE.2019.00033.

[Lar+20] Lars Larsson, William Tärneberg, Cristian Klein, Erik Elmroth, and Maria Kihl. "Impact of etcd deployment on Kubernetes, Istio, and application performance".
In: *Software: Practice and Experience* 50.10 (2020), pp. 1986–2007.
DOI: 10.1002/spe.2885.

[Ma+20] Yehan Ma, Chenyang Lu, Bruno Sinopoli, and Shen Zeng.
"Exploring Edge Computing for Multi-Tier Industrial Control".
In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (11 Nov. 2020), pp. 3506–3518. ISSN: 0278-0070.
DOI: 10.1109/TCAD.2020.3012648.
URL: https://ieeexplore.ieee.org/document/9211472/.

[SLM20] Darius Saif, Chung-Horng Lung, and Ashraf Matrawy.
"An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse".
In: *IEEE International Conference on Communications* (2020).
ISSN: 15503607. DOI: 10.1109/ICC42927.2021.9500258.

[Ska+20] Per Skarin, William Tärneberg, Karl-Erik Arzen, and Maria Kihl. "Control-over-the-cloud: A performance study for cloud-native, critical control systems". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE. IEEE, Dec. 2020, pp. 57–66. ISBN: 978-0-7381-2394-3. DOI: 10.1109/UCC48980.2020.00025.

[Zha+20] Qi Zhang, Lin Gui, Fen Hou, Jiacheng Chen, Shichao Zhu, and Feng Tian. "Dynamic Task Offloading and Resource Allocation for Mobile-Edge Computing in Dense Cloud RAN". In: *IEEE Internet of Things Journal* 7.4 (Apr. 2020), pp. 3282–3299. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2967502.

[DJ21] Jasenka Dizdarevic and Admela Jukan. "Experimental Benchmarking of HTTP/QUIC Protocol in IoT Cloud/Edge Continuum". In: *ICC 2021 - IEEE International Conference on Communications*. IEEE, 2021. ISBN: 978-1-7281-7122-7. DOI: 10.1109/ICC42927.2021.9500675.

[IS21] Jana Iyengar and Ian Swett. *QUIC Loss Detection and Congestion Control*. RFC 9002. May 2021. DOI: 10.17487/RFC9002.

[IT21] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000.

[Moo+21] Nazanin Moosavi, Mahnaz Sinaie, Paeiz Azmi, and Jyrki Huusko. "Delay Aware Resource Allocation With Radio Remote Head Cooperation in User-Centric C-RAN". In: *IEEE Communications Letters* 25 (7 July 2021), pp. 2343–2347. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2021.3069235. URL: https://ieeexplore.ieee.org/document/9388706/.

[Mor21] Farnaz Moradi. "Improving DRX Performance For Emerging Use Cases In 5G". English. PhD thesis. 2021.

[Bis22] Mike Bishop. *HTTP/3*. RFC 9114. June 2022. DOI: 10.17487/RFC9114.

[Fil+22] Abderrahime Filali, Zoubeir Mlika, Soumaya Cherkaoui, and Abdellatif Kobbane. "Dynamic SDN-Based Radio Access Network Slicing With Deep Reinforcement Learning for URLLC and eMBB Services". In: *IEEE Transactions on Network Science and Engineering* 9 (4 2022), pp. 2174–2187. ISSN: 23274697. DOI: 10.1109/TNSE.2022.3157274.

[ITC22] Amjad Iqbal, Mau-Luen Tham, and Yoong Choon Chang. "Resource allocation for joint energy and spectral efficiency in cloud radio access network based on deep reinforcement learning". In: *Transactions on Emerging Telecommunications Technologies* 33 (4 Apr. 2022). ISSN: 2161-3915. DOI: 10.1002/ett.4417. URL: https://onlinelibrary.wiley.com/doi/10.1002/ett.4417.

[Liu22] Guo-Ping Liu. "Coordinated Control of Networked Multiagent Systems via Distributed Cloud Computing Using Multistep State Predictors". In: *IEEE Transactions on Cybernetics* 52 (2 Feb. 2022), pp. 810–820. ISSN: 2168-2267. DOI: 10.1109/TCYB.2020.2985043. URL: https://ieeexplore.ieee.org/document/9082811/.

[Sha+22] Mahdi Sharara, Sahar Hoteit, Veronique Veque, and Francesca Bassi. "Minimizing Power Consumption by Joint Radio and Computing Resource Allocation in Cloud-Ran". In: vol. 2022-June. IEEE, June 2022, pp. 1–6. ISBN: 978-1-6654-9792-3. DOI: 10.1109/ISCC55528.2022.9912943. URL: https://ieeexplore.ieee.org/document/9912943/.

[Zav22] Dmitry Zavalishin. *MQTT/UDP Documentation, Release 0.5-0*. Last Accessed: August 4, 2023. May 2022. URL: https://buildmedia.readthedocs.org/media/pdf/mqtt-udp/latest/mqtt-udp.pdf.

[Zha+22] Yongnan Zhang, Yonghua Zhou, Huapu Lu, and Hamido Fujita. "Spark Cloud-Based Parallel Computing for Traffic Network Flow Predictive Control Using Non-Analytical Predictive Model". In: *IEEE Transactions on Intelligent Transportation Systems* 23 (7 July 2022), pp. 7708–7720. ISSN: 1524-9050. DOI: 10.1109/TITS.2021.3071862.

[Dai+23] Li Dai, Yaling Ma, Runze Gao, Jinxian Wu, and Yuanqing Xia. "Cloud-Based Computational Model Predictive Control Using a Parallel Multiblock ADMM Approach". In: *IEEE Internet of Things Journal* 10 (12 June 2023), pp. 10326–10343. ISSN: 23274662. DOI: 10.1109/JIOT.2023.3238508.

[Liu+23] Bo Liu, Pengcheng Zhu, Jiamin Li, Dongming Wang, and Xiaohu You. "Energy-Efficient Optimization in Distributed Massive MIMO Systems for Slicing eMBB and URLLC Services". In: *IEEE Transactions on Vehicular Technology* 72 (8 Aug. 2023), pp. 10473–10487. ISSN: 0018-9545. DOI: 10.1109/TVT.2023.3260988. URL: https://ieeexplore.ieee.org/document/10079135/.

[Nok23] Nokia. *Mining and mission-critical wireless connectivity: Laying the foundation for the digital transformation of mining*. Tech. rep. Last Accessed: November 1, 2023. Nokia, 2023. URL: https://www.nokia.com/networks/industries/mining/#bringing-private-wireless.

[Oca+23] Andres F. Ocampo, Mah-Rukh Fida, Juan F. Botero, Ahmed Elmokashfi, and Haakon Bryhni. "Opportunistic CPU Sharing in Mobile Edge Computing Deploying the Cloud-RAN". In: *IEEE Transactions on Network and Service Management* (2023), pp. 1–1. ISSN: 1932-4537. DOI: 10.1109/TNSM.2023.3304067. URL: https://ieeexplore.ieee.org/document/10214346/.

[SM23] Darius Saif and Ashraf Matrawy. "An Experimental Investigation of Tuning QUIC-Based Publish-Subscribe Architectures in IoT". In: arXiv, 2023. eprint: 2208.11178 (cs.NI).

[Yan+23] Dandan Yan, Benjamin K. Ng, Wei Ke, and Chan-Tong Lam. "Deep Reinforcement Learning Based Resource Allocation for Network Slicing With Massive MIMO". In: *IEEE Access* 11 (2023), pp. 75899–75911. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3296851. URL: https://ieeexplore.ieee.org/document/10186882/.

[AG] Deutsche Messe AG. *HANNOVER MESSE Events*. Last accessed Nov. 2020. URL: https://www.hannovermesse.de/en/.

[Arc] HTTP Archive. *Report: State of the Web*. Last Accessed: August 3, 2023. URL: https://httparchive.org/reports/state-of-the-web?start=2018%5C%5F03%5C%5F01&end=latest&view=list.

[Goo] Google. *Google Edge Network*. Last Accessed: November 9, 2023. URL: https://peering.google.com/%5C#/learn-more/quic.

[Hat] Red Hat. *Evolution to a 5G core network*. Last Accessed: August 10, 2023. URL: https://www.redhat.com/en/topics/5g-networks/evolution-to-a-5g-core.

[JC] Matt Joras and Yang Chi. *How Facebook is bringing QUIC to billions*. Last Accessed: November 9, 2023. URL: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[Ric] Felix Richter. *Cloud Infrastructure Market: Amazon Maintains Lead in the Cloud Market*. Last Accessed: August 11, 2023. URL: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[ST] Per Skarin and William Tärneberg. *cotc-modsim*. Last Accessed: November 29, 2022. URL: https://www.thefuturenow.se/pypi/cotc-modsim/.

[Stå] Jon Stålhammar. *Massive MIMO Simulator*. Github. URL: https://github.com/jost95/massive-mimo-simulator.

[Web] Web3Techs. *Usage statistics of Default protocol https for websites*. Last Accessed: August 3, 2023. URL: https://w3techs.com/technologies/details/ce-httpsdefault.

Popular Science

In today's digital age, "the cloud" is no longer just a fluffy white thing in the sky—it's a revolutionary technology that's changed the way we live and work. At its core, the cloud is a shared pool of computing resources. Think of it like a public library for computers; instead of buying every book you want to read, you can borrow it from the library and return it when you're done. This "pay-as-you-go" model has made it affordable for businesses and individuals to use complex software and store massive amounts of data without the need for owning sophisticated hardware.

People use the cloud in their daily lives, often without realizing it. Think of Google Docs or Microsoft's online Office suite. These are prime examples of "Software-as-a-Service" in cloud computing, where you don't need to install bulky software; you just use it directly online. Gamers, too, can enjoy the cloud with platforms like Xbox Cloud Gaming, which lets you play games without the need for a dedicated console.

But while the cloud offers convenience, it doesn't always deliver the best user experience. Imagine you're writing in Google Docs, and it lags, especially with larger files. Or you're in the middle of a fast-paced Xbox game, and you experience delays or lower visual quality. That's like streaming a film and having it pause or blur at the most critical moments.

But what happens when this technology enters complex systems like industrial facilities? Welcome to the realm of cloud integrated systems, the focus of my research. These systems take the concept of the cloud and apply it to industries where speed, reliability, and accuracy are paramount. In these settings, moving parts of the computational workload to the cloud could be a game-changer but also a potential headache.

In my research, I specifically tackle two kinds cloud integrated systems: Cloud RAN and Cloud Control Systems. In a nutshell, Cloud RAN shifts some of a traditional radio base station's heavy lifting to the cloud. Similarly, a cloud control system takes part of a cyber-physical system—like a manufacturing line—and moves its computational tasks to the cloud.

Why does this matter? Because when you move things to the cloud, you inherit all of its challenges. The industrial systems have stringent performance requirements. They can't afford to wait for the cloud's slowpoke tendencies. They need their data now, not a split second later, to keep operations running smoothly.

My research aims to answer two fundamental questions. First, can we successfully

integrate these industrial systems with the cloud? And if so, how can we make it better, accepting that the cloud and network communications will never be perfect?

I've identified two main culprits behind these issues: the shared nature of the cloud and the latency due to distance. The cloud is like a communal pie—when you share it with others, you don't get as much as you'd like, and you have to wait your turn. Similarly, the further the pie (or cloud) is from you, the longer it takes for a piece to reach your plate.

But here's where my work comes in. Instead of trying to make the cloud or networks faster—which is often beyond our control—I look at optimizing our own systems to adapt to these limitations. I work on solutions from the system's perspective to mitigate these challenges, ensuring that despite the imperfections in the cloud and network, our cloud integrated systems can still run as efficiently as possible.