Integration of Clouds to Industrial Communication Networks

Peng, Haorui

2021

Integration of Clouds to Industrial Communication Networks

—

Haorui Peng

Lund 2021

Department of Electrical and Information Technology
Lund University
Box 118, SE-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 10pt
with the LaTeX Documentation System

Series of licentiate and doctoral theses
No. 136
ISSN 1654-790X
ISBN 978-91-7895-725-5 (printed)
ISBN 978-91-7895-726-2 (electronic)

Printed in Sweden by *Tryckeriet i E-huset*, Lund.
January 2021.

Abstract

Cloud computing, owing to its ubiquitousness, scalability and on-demand access, has transformed into many traditional sectors, such as telecommunication and manufacturing production. As the Fifth Generation Wireless Specifications (5G) emerges, the demand on ubiquitous and re-configurable computing resources for handling tremendous traffic from omnipresent mobile devices has been put forward. And therein lies the adaption of cloud-native model in service delivery of telecommunication networks. However, it takes phased approaches to successfully transform the traditional Telco infrastructure to a softwarized model, especially for Radio Access Networks (RANs), which, as of now, mostly relies on purpose-built Digital Signal Processors (DSPs) for computing and processing tasks.

On the other hand, Industry 4.0 is leading the digital transformation in manufacturing sectors, wherein the industrial networks is evolving towards wireless connectivity and the automation process managements are shifting to clouds. However, such integration may introduce unwanted disturbances to critical industrial automation processes. This leads to challenges to guarantee the performance of critical applications under the integration of different systems.

In the work presented in this thesis, we mainly explore the feasibility of integrating wireless communication, industrial networks and cloud computing. We have mainly investigated the delay-inhibited challenges and the performance impacts of using cloud-native models for critical applications. We design a solution, targeting at diminishing the performance degradation caused by the integration of cloud computing.

Preface

This licenciate thesis concludes my work as a licentiate candidate, and is comprised of two parts. The first part gives an overview of the research field in which I have been working during my licentiate and a brief summary of my contribution in it. The second part is composed of four included papers that constitute my main scientific work:

1. Haorui Peng, Emma Fitzgerald, William Tärneberg, Maria Kihl, '5G radio access network slicing in massive MIMO systems for industrial applications', in *2020 Seventh International Conference on Software Defined Systems (SDS)*, Paris, France, Jul 2020.

2. Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl, 'Massive MIMO pilot scheduling over Cloud RAN for Industry 4.0', in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Croatia, Sept. 2020.

3. Haorui Peng, William Tärneberg, Maria Kihl, 'Latency-aware radio resource scheduling over Cloud RAN for Industry 4.0', submitted

4. Johan Ruuskanen, Haorui Peng, Alfred Åkesson, Lars Larsson, Maria Kihl, 'FedApp: a research sandbox for application orchestration with Kubernetes on the next-generation cloud and edge computing infrastructures', to be submitted

During my licentiate, I have also contributed to the following publications, which are not included in the thesis:

1. Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl, 'Massive MIMO pilot scheduling over Cloud RAN', in *16th Swedish National Computer Networking Workshop (SNCNW 2020)*, Kristianstad, Sweden, May 2020.

2. Johan Ruuskanen, Haorui Peng, Alexandre Martins, 'Latency prediction in 5G for control with deadtime compensation' in *IoT-Fog '19 Proceedings of the Workshop on Fog Computing and the IoT*, Montreal, Canada, Apr 2019.

Acknowledgements

First and foremost, I want to express my deepest gratitude to main supervisor, Maria Kihl, for her guidance, patience and all the insightful feedbacks in my work. Maria has given me persistent support along the journey of my PhD and lead me to be an independent researcher.

I'm greatly indebted to my co-supervisor, William Tärneberg, for all his help and encouragement since the first day I started my PhD. I couldn't manage to surmount all the difficulties on my research path without all the discussions and inspirations from him. I'm also grateful to Emma Fitzgerald, also my co-supervisor. Her talent and enthusiasm have inspired me a lot in my work.

I wish to thank all the collaborators and the ones have helped me in my projects, big and small, which are all the stepping stones on my research journey. A special thanks to Lars Larsson, who also supervised me in one of my projects. His thoughtfulness, encouragement and invaluable experiences in my research filed are of great help.

Big thanks to all my friends and colleagues in Broadband Communication group and Networking group. I have enjoyed all your company and I really appreciate all your supports and comforts at my difficult times.

Last but not least, I would like to acknowledge the great love from my family. Thank you for understanding and supporting every decision and choice I have made. You are the ones who kept me going on steadily. Profoundly miss my dearest grandfather, Qingxiang Cheng.

Haorui Peng

Contents

Part I

Overview of Research Field

Chapter 1

Introduction

Advanced by widely spreading Internet services and enabled by maturing virtualisation and containerisation technology, cloud computing has became the primary trend of utility computing – a concept brought in by John McCarthy in 1961, predicting "using computing as public utility, just as the telephone system" [1]. A brief idea of cloud computing is to centralise the computing resources in large-scaled data centres. But the concept of "cloud" is more than how people may picture a data centre from a traditional standpoint, wherein numerous applications are designed to run on dedicated hardware infrastructure and operation systems.

The US National Institute of Standards and Technology (NIST) defines cloud computing as follows:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [2].

During recent years, Software Defined Networks (SDN) and Network Function Virtualisation (NFV) are playing key roles to steer the transformation of 5G networks towards a service-base architecture [3]. The 5G Infrastructure Public Private Partnership (5GPPP) envisions that 5G integrates networking, computing and storage into one high-capacity, programmable and unified infrastructure [4]. This transformation and integration of Information Technology (IT) have directed the telecommunication industry to the journey of cloud computing and enforces the cloud-native delivery model to design 5G applications [5].

Originated from order of magnitude increasing Internet of Things (IoT) devices and mobile applications nowadays, the enormous size of data has put growing bandwidth pressure over the links to the centralised cloud. Therefore, edge computing, a new paradigm of cloud computing was brought on stage, which provides computing resources with geographically distributed, small-scale data centres located at the edge of networks. The most distinct examples of exploiting edge computing in the Telco sector are Mobile Edge Cloud (MEC) and Cloud RAN (Phase 2). MEC, also as a key enabler for 5G networks, specifically refers to the computing site operating at the edge of RANs, and, as defined by European Telecommunications Standards Institute (ETSI), is in close proximity to mobile subscribers [6]. Cloud RAN is a candidate architecture for future Radio Base Stations (RBSs). The commonality and differentiation of these two paradigms are described in Chapter 2.

In this thesis, we have investigated the performance challenges in the integration of edge cloud and network systems, mainly but not limited to RANs under industrial scenarios. We started with examining the feasibility of wireless network establishment for industrial applications, from the perspective of the end-user performances (Paper I), and continued by investigating the impacts and challenges of cloud integration to this network system (Paper II and III). Then we expanded our interests to the deployment of cloud federation, by addressing the network limitations among cloud infrastructures that are nation-widely distributed (Paper IV).

The scope of my work in this thesis with respect to the distance between services/functions and their end-users, are; Firstly co-located (Paper I), then separated by tens of kilometres (Paper II and III), and eventually by nationwide distance (Paper IV). But it has to be noted that the geographic distance is not the only (even not the most important) factor that introduces the network limitations and impacts on end-user performances in our addressed systems. We will briefly introduce the challenges for deploying the considered network systems in Chapter 3.

This thesis is structured as follows; In Chapter 2, a brief overview on the background of the included papers is presented, which is twofold: 1) The industrial wireless network system and its performance requirements. 2) Cloud integration operation for a network system (industrial and beyond). Chapter 3 provides our vision on the challenges of the described system deployment, which are addressed by included papers summarised in Chapter 4. In the end of the same chapter we make a conclusion on how the work in each paper continued from the previous one, as well as our planned work following the research scope presented in this thesis.

Chapter 2

Background Overview

In this chapter we briefly introduce the research topics covered in this thesis. The target systems addressed by each of our paper has considered at least one of the aspects described in this chapter. We will also sketch our vision on "cloud integrated network system" and explain how it is tied with the topics outlined in this chapter.

2.1 Industrial Automation and Network

In this section, we present the considered industrial applications and their traffic characterises, which determine a target scenario and performance model of the network systems addressed by most of the included papers.

2.1.1 Industry 4.0 and IIoT

In 2011, the concept of Industry 4.0 (also termed as the Fourth Industrial Revolution afterwords) was introduced in Hannover Trade Fair [7] for the future development of German manufacturing industry. Industry 4.0 represents the digital transformation of manufacturing production, through integrating the traditional production systems with emerging technologies including IoT, 5G, Artificial Intelligence (AI), cloud computing, et cetera.

IoT plays a significant role in the era of Industry 4.0. One definition of IoT is given as:

> Every object or "thing" is embedded with a sensor and is capable of automatically communicating its state with other objects and automated systems within the environment [8].

Table 2.1: Standardised SST values. From [3].

Slice/Service type	**SST value**	**Characteristics**
eMBB	1	Slice suitable for the handling of 5G enhanced Mobile Broadband.
URLLC	2	Slice suitable for the handling of ultra-reliable low latency communications.
MIoT (or mMTC)	3	Slice suitable for the handling of massive IoT.
V2X	4	Slice suitable for the handling of V2X services.

A simple definition of Industrial Internet of Things (IIoT) could then refer to making use of IoT in manufacturing. In [9], a more detailed definition of IIoT is provided by appealing all the potential technologies that compose it, including networked smart objects, autonomous communications and optional cloud and edge computing.

Following the 3rd Generation Partnership Project (3GPP) Standardised Slice/Service Type (SST) values in Table 2.1, the IoT devices and applications can be identified and categorised by their commonality amongst the network services types.

In the famous three principle dimension figure of 5G usage scenarios envisioned in IMT-2020 [10], the communication characteristics of industrial automation applications are labelled in between the Ultra-Reliable and Low-Latency Communication (URLLC) and Massive Machine Type Communication (mMTC) type, as shown in Figure 2.1.

Whilst unlike the extreme user plan requirements in 5G URLLC services[11], which are 1ms latency and $1 - 10^{-5}$ successful transmission probability, the latency and reliability of industrial IIoTs requirements are also dependent on the properties of the manufacturing process. Such as concluded by [12], the tracking devices connected to the network only requires best-effort latency and low availability. This kind of devices bring the end-to-end visibility to the supply chain and belongs to type 3 in Table 2.1. The smart sensors and actuators embedded in the manufacturing equipment are categorised as type 2 or type 1, they ask for low latency less than 10ms and high availability larger than 95%. The latter will be the major component framing the traffic to our systems in this thesis.

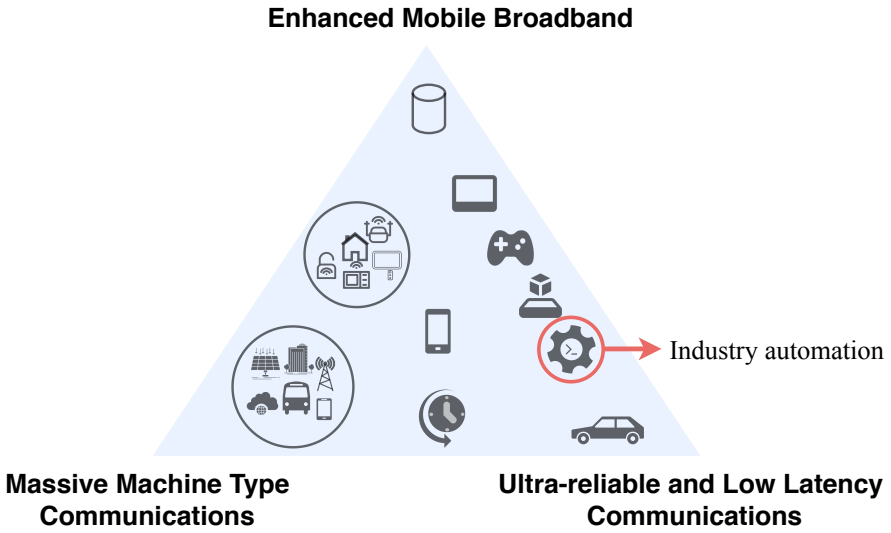


Figure 2.1: Usage scenarios of IMT for 2020 and beyond.

2.1.2 Evolving towards Wireless Connectivity

The magnitude of the demands on information exchange from tremendously increasing industrial applications stimulates the evolution of industrial communication networks over years. Industrial networks should be capable of coping with the automation specific requirements like deterministic, reliable and efficient communications. The traditional solution for industrial automation networks in the 20th century was cable networks, such as PROFIBUS, Controller Area Network (CAN) or INTERBUS. Real-time Ethernet solutions has also emerged along with the prevailing Internet technologies [13]. Industry 4.0 appeals for a more flexible and ubiquitous Internet connectivity for smart IoT devices and distributed production organisation, which advanced the adoption of wireless technology and mobile Internet in industrial automation. Considerable advantages of the wireless network such as ease of installation and low maintenance cost also strengthen its competitiveness in the era of Industry 4.0.

It is widely recognised that 5G will occupy an important place in the future industrial automation arena, as it promises the same level of reliability, capacity and latency as wired networks [13, 14, 15], which however, remain to be challenges for most of the current wireless standards. These fascinating features of 5G are furnished by the enabling technologies such as massive Multiple Input Multiple Output (MIMO), network slicing and milimetre Wave (mmWave) [16].

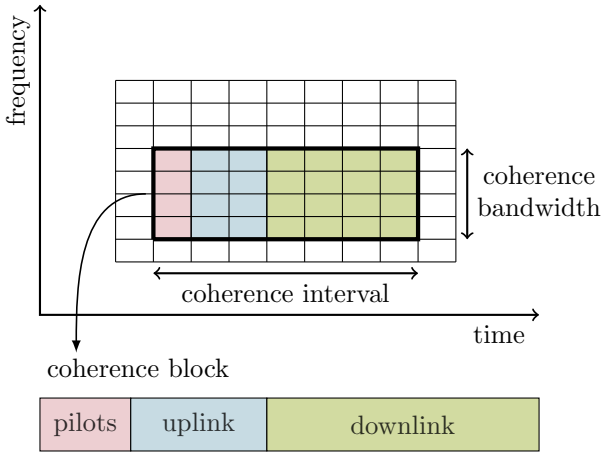


Figure 2.2: Example of massive MIMO time-frequency spectrum of one coherence interval.

Pilot Access of Massive MIMO

The new paradigm of massive MIMO antenna system in the 5G era could provide robust radio access for the end-users with energy and spectrum efficiency. Making use of large-scale antennas (more than number of end-users), massive MIMO promises to handle orders of magnitude more data traffic and simultaneously serve numerous end-users under the same time-frequency spectrum [17]. With such features it offers the opportunity of integrating efficient radio access for communication of massive IIoTs into industrial automation network.

For radio access networks, Channel State Information (CSI) acquisition in each coherence block through pilot sequences is necessary for a RBS, especially in crowded scenarios[18]. An example of time-frequency spectrum of massive MIMO is shown as Figure 2.2 and explained in [19].

The number of orthogonal pilots are limited by the channel coherence interval, but the number of end-users in massive IIoT use cases are considerably larger than number of available pilots in a coherence interval even for massive MIMO. It is non-trivial to choose suitable pilot access methods by considering traffic specifics under different scenarios to avoid pilot collision or contamination. For instance, random access pilots are proposed by [20] for mMTC traffic and an optimised device scheduling strategy is studied in [21] for alert traffic in industrial automation network.

2.1.3 Cloud Assisted Industrial Process

Since on-board processing are restricted on the IoT devices, offloading the data analysis and computing originated from large-scale deployment of IIoT applications to clouds is acknowledged the most competitive solution for manufacturing sectors [22, 23, 24], given the advantageous of "inifinte" computing resources and storage capacity. It has been reported achieving productivity improvements in manufacturing plants via integrating IoT applications with cloud services [25].

However, it introduces a bottleneck on the network bandwidth and unpredictable delays while deploying services for numerous IIoT devices in the centralised cloud. Therefore, edge computing is widely accepted as the open platform to provide computing, storage and networks services, especially for mission-critical applications in industrial automation. By shifting computing to the vicinity of the manufacturing plant, edge cloud significantly improves the response time of cloud services and can better achieve the low latency requirement for real-time processing [26, 27, 28]. The case study in [29] has shown that an integration of edge cloud with local controller provides a considerable performance improvement in a multi-tier industrial control system.

2.2 Cloud Integrated Network System

As typical examples that exploit cloud technologies at the access network site of the Telco world, both Cloud RAN and MEC are shining a spotlight in the 5G era and beyond. Both paradigms, can be viewed as "cloud integrated network system", but there are distinct differences between the two terms. In this section we first take a close look at the two systems and then provide our view on "cloud integrated network system".

2.2.1 Mobile Edge Cloud

As of today, there is no well-established MEC infrastructure by Mobile Network Operators (MNOs), even though MEC is broadly accepted being in the path of 5G RAN evolution. In this thesis, we refer the definition of MEC to the standardisation driven by ETSI [6] and acknowledge several features in the following of this section.

MEC aims at providing IT service environment at close proximity of mobile subscribers, so that the endpoint of the MEC-enabled applications are shifted to the edge of the network from the centralised cloud, to which the traffic needs to traverse across the Core Network (CN) and Internet. Complementary to the advantages of cloud computing, MEC is equipped with characterises of low

latency, cost efficient and real-time processing, which opens up the opportunity for MNOs to delivery mission-critical services for subscribers.

The "killer app" of MEC could be smart phone applications and content, Augmented Reality (AR) applications, connected cars, IoT applications and industrial automation (use case in Section 2.1.3 for example). A MEC not only plays a role of an IT cloud at the edge, which builds on application centric environments. It is also promises to provide deployment, orchestration and management for Virtual Network Function (VNF) as a Telco cloud, which makes use of NFV architectures and requires higher bandwidth connections [30].

2.2.2 Cloud RAN

The concept of Cloud RAN grew out of the centralised RAN, which separates Remote Radio Heads (RRHs) and Base-Band processing Units (BBUs) of RBSs and processes the base-band signals in a centralised base station [31]. But in centralised RAN, the resources are not pooled or virtualised at the central base station. Cloud RAN has taken one step further, which takes the initiative of base-band resource pool and performs joint signal processing, by taking advantages of Software Defined Radio (SDR). The second phase of Cloud RAN aims at the BBU pool evolution, by leveraging virtualisation technologies, making use of General-purpose Processors (GPPs) instead of DSPs and approaching towards real-time base-band processing in the fashion of cloud computing. In this thesis, we address the second phase of Cloud RANs, considering that part of the signal processing function chain is deployed in a cloud-native fashion on GPP components.

All things considered, Cloud RAN is a candidate architecture for future RBSs, in which the signal processing functions are partially or fully deployed in a centralised BBU pool building on virtualisation and cloud technologies. But the "cloudified" BBU pool can be differentiated from MEC by the functions they host. A BBU pool is responsible for base-band processing functions such as modulation, channel coding, radio resource mapping and Medium Access Control Layer (MAC) functions, whereas a MEC covers a MNO's own applications and VNFs.

2.2.3 Our Definition of Cloud Integrated Network System

Nevertheless, the MEC and Cloud RAN are not mutually excluded, but are highly complementary technologies. Additionally, it significantly mitigates the cost of deployment and maintenance by having a co-located Cloud RAN and MEC infrastructure than standalone sites of both [32]. The workloads of a

Cloud RAN can be long lived and have critical requirements as it supports radio signal processing, which yields the provision of more powerful, sustainable and even redundant computing resources. MECs can more easily scale up and down the computing units to support short lived workloads.

Moreover, NFV is believed to facilitate Cloud RAN deployments. Likewise, there is high potential that MEC will utilise NFV architecture in the long run to orchestrate and manage its services and applications. The emerging NFV technology provides a great opportunity for the convergence of MEC and Cloud RAN. The co-deployment of the two infrastructures is currently under exploration and investigation by ETSI, who has proposed several RAN services that can be exposed to MEC for achieving service optimisation at the Cloud RAN [32].

In this thesis, the cloud integrated network system we address is defined as follows; A distributed system in which all units are network connected, and which shifts the computing units to a cloud site that builds on a pool of virtualised computing, networking and storage components, and makes use of cloud-native techniques to orchestrate, manage and execute the services running on it.

The previously mentioned systems in Section 2.1.3 and Cloud RAN are both deemed to be cloud integrated network systems in this thesis, as well as MEC, if we take into account the mobile end-users accessing the services hosted at the edge through radio access network. We note that the term Cloud RAN is used in PAPER II and III, because we have specifically considered a MAC layer function (pilot access scheduling described in Section 2.1.2) hosted in the cloud, which is a typical RAN service that is executed by the BBU of a RBS.

Chapter 3

Challenges of Cloud Integrated Wireless Industrial Network

3.1 Heterogeneous Traffic and Pilot Shortage

Although massive MIMO has increase the capacity for over 10 times and is capable to handle order of magnitude increasing data traffic for 5G RAN, it may still ran into pilot shortage problems, especially in a mMTC scenario, where the number of orthogonal pilots sequences are much smaller than the number of end-users [18].

Under an industrial automation scenario, when heterogeneous IIoT devices with different Quality of Service (QoS) requirements are connected by a single-cell wireless network, the pilot access problem may become more intricate, since different type of applications may need different pilot access protocols. For example, considering the co-existence of URLLC and mMTC devices in the same access network. Pilot pre-allocation will be more suitable for URLLC devices to prevent pilot collision and guarantee persistent and reliable transmissions. However for mMTC devices, which generally have sporadic transmission demands, a random access protocol could be a more viable solution as suggested in [18, 20, 33].

In Paper I, we took into account this heterogeneous industrial scenario and investigated the feasibility to accommodate the two types of traffic in a single-cell massive MIMO radio access network by making use of the network slicing concept in the radio spectrum, so that a dedicated pilot allocation strategy can

be deployed for each category of end-users in the scenario.

3.2 Cloud Execution Environment

Cloud execution environment is usually non-deterministic and unpredictable, which is contrary to the low-latency and ultra-reliable characteristics provided by 5G and required from industrial automation networks.

The computing resources in cloud are pooled to serve the customers with a multi-tenant model, but can introduce uncertainties to computing performances. Different from applications running on traditional servers, each owns a single physical server independently, a cloud service customer has no control on the location of the provided resources, neither the knowledge of other tenants sharing the same physical resource.

Resource pooling are achieved by virtualisation, which is the key technology leveraged by cloud computing. Virtualisation benefits the modern data centres with higher flexibility, faster resource provision, cost reduction and higher resource utilisation, however, at the cost of performance degradation. As it adds abstraction layers on top of physical machine, yielding a longer path of the workloads than that in a bare-metal server. Researches and experiments have shown performance degradation on I/O, networking and memory access with virtualisation [34, 35, 36]. All this would lead to, from the perspective of a service user, a longer and uncertain response time from the application.

Besides the virtualisation technology, the cloud-native deployment also brings overheads to the application performance. We hereby take into account an application running on a standalone Docker[1] container. A container is the smallest necessary computing unit in cloud-native deployment and it shares the Linux kernel with the host machine. The inbound/outbound traffic of a container is forwarded by a software bridge. A container is unaware of itself running on an abstraction layer, and sees itself as a normal machine with full network stack. Thus the host machine of a Docker container needs to encapsulate incoming packets with IP headers in order to direct them to the destined container. This overall brings overheads on performances in terms of networking by adding extra layers on the path of workloads.

We consider Kubernetes[2] deployment, which is a typical and the most popular orchestration platform for container deployment, management and scaling. Briefly speaking, Kubernetes wraps containerised applications into Pods on Nodes, where a Node is a virtual machine or a physical machine. When

[1] `https://www.docker.com`
[2] `https://kubernetes.io`

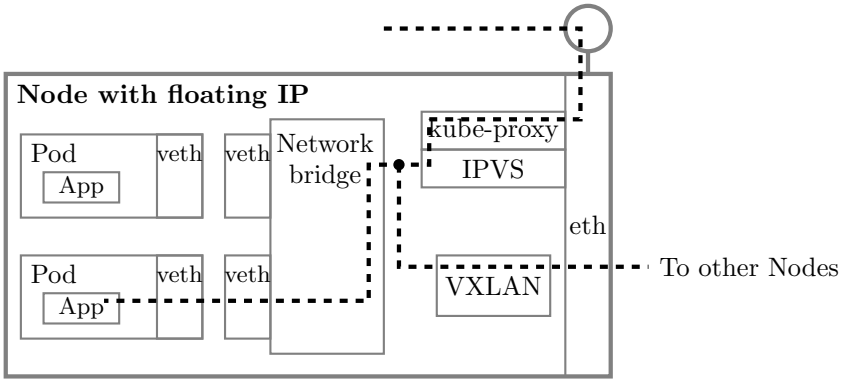


Figure 3.1: An example of the workload path to reach the service in a typical cloud environment through Kubernetes NodePort deployment.

deploying a cloud service on top of Kubernetes, where the networking functionalities are provided by Container Network Interface (CNI), more hops are added along the path of workloads to reach the end-point of a service. In [37], the authors gave an example on how the workload traffic is directed to the service end-point in a Kubernetes node when having a NodePort[3] type of service, as is it shown in Figure 3.1. Additionally, if the service Pod is deployed in another Node of the cluster but without floating IP, the workload traffic will be directed over the Virtual Extensible LAN (VXLAN) to the other Node. An overview of networking in a native Kubernetes deployment is also given in [37].

We show in Figure 3.2 our measurements on response time of a simple "UDP ping" application when being deployed in (a) a bare-metal cluster in the same Local Area Network (LAN) with the client machine, and (b) when being deployed in a cloud date centre residing in the same city. Under each scenario, we compared the application as running in a host machine, in a stand-alone Docker container and in a Kubernetes Pod. As depicted in Figure 3.2, for the same application, the mean response time and its variances have increased as it is running on a host, a docker container and a Pod in Kubernetes cluster. We also see that the Kubernetes deployment in a cloud data centre has more overheads than in a bare-metal cluster, as we don't know if the Pod is hosted by the node with floating IP. If the application is hosted by a node with only private IP address, the workload will be re-directed over a VXLAN to the end-point.

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

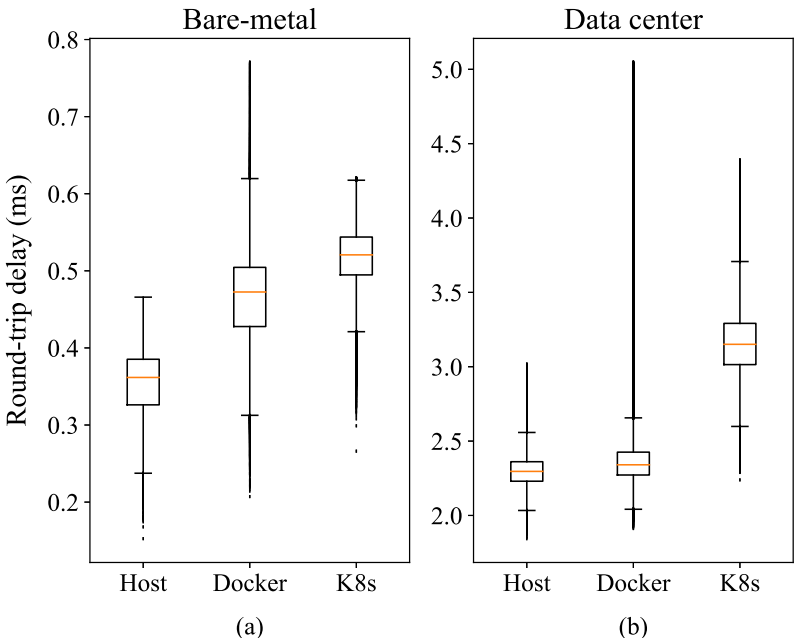


Figure 3.2: UDP latency measurements on response times to different type of services deployments, 1.5% outliers removed

In short, introducing the cloud deployment for mission-critical applications would have more challenges in term of user experience, as extra abstraction layers are added to the path of the workload traffic. In Paper II, considering more delays is added by the cloud deployment, we show that it is feasible to integrate cloud as computing unit for a critical RAN service (the pilot access function described in Section 2.1.2). In Paper III we propose a strategy that mitigates the impacts of delays introduced by a cloud deployment.

Chapter 4

Summary and Contributions

4.1 Research contributions

The four papers included in this thesis are summarised below, which illustrates the path of our investigation on a cloud-integrated network system. This chapter gives an overview on the content of each paper and detail my main contributions in each work.

4.1.1 Paper I: 5G Radio Access Network Slicing in Massive MIMO Systems for Industrial Applications

This paper investigates the feasibility of accommodating two different types of traffic in a single-cell RAN supported by massive MIMO. Two types of end-users sharing the same radio spectrum under an industrial scenario are addressed. The traffic from the two groups of end-users have characteristics of URLLC and mMTC.

The network slicing concepts are adopted to tailor the time-frequency spectrum of massive MIMO into two slices for different types of end-users. A two-level MAC scheduling scheme is proposed. First, the scheme split the spectrum resource into two slices and second, it assigns radio resources to individual end-users under each slice. In this way, different types of end-users can be treated separately and various demands can be met in a single-cell. In the paper, different schedulers to assign the resources are compared, in order to investigate suitable scheduling algorithms for numbers of end-user requirements.

I was the primary researcher in this work and my contribution is in system definition and modelling, simulation development, performance evaluation and analysis.

4.1.2 Paper II: Massive MIMO Pilot Scheduling over Cloud RAN for Industry 4.0

In this paper, the feasibility of deploying Cloud RAN system under a industrial scenario was investigated. The Cloud RAN integrates cloud as the processing unit of the RAN system. The cloud hosts a MAC layer network service for massive MIMO, which remotely schedules radio resources for the end-users. The paper focused on the latency and availability requirements for the critical applications that are involved in control process of manufacturing production systems. As the remote scheduler and the actuation of the scheduling are hosted by two far apart separated units (the BBU pool and RRH), and the execution environment in the cloud is also challenging, the paper investigated if the Cloud RAN can meet the industrial criteria of critical applications, by considering the latency limitations incurred in the system.

For system evaluation, a commonly deployed Earlist Deadline First (EDF) scheduling algorithm was deployed for different scenarios regarding end-user mobility, density and transmission deadlines. Two performance metrics were defined and the paper summarised the operational space within which the Cloud RAN is feasible to be deployed.

I was the primary researcher in this work and my contribution is in system definition and modelling, simulation development, experiment setup, performance evaluation and analysis.

4.1.3 Paper III: Latency-aware Radio Resource Scheduling over Cloud RAN for Industry 4.0

This paper is built on the scenario and system model of Paper II. But in addition to the critical applications, the paper also takes into account the non-critical applications under the same scenario, since these applications also have demands for radio resources. Following the discoveries in Paper II, enormous amount of resources are wasted due to redundant allocations to the critical applications in order to meet their latency requirements. This resource waste will cause resource starvation and eventually transmission failure on the non-critical applications.

Furthermore, in the simulation model, not only the length of the delays incurred by Cloud RAN system were taken into account, but also the stochastic

characteristics of the delays, which are caused by the uncertainties in the cloud execution environment.

A novel scheduling strategy was proposed. The main aim of the strategy is to avoid resource waste so that the network system can accommodate both critical and non-critical applications simultaneously. The results showed that this strategy can significantly improve the utilisation performance, without comprising the transmission reliability of the end-users.

I was the primary researcher in this work and my contribution is in system definition and modelling, solution design, simulation development, performance evaluation and analysis.

4.1.4 Paper IV: FedApp: a Research Sandbox for Application Orchestration with Kubernetes on the Next-Generation Cloud and Edge Computing Infrastructures

This paper presents the experience of building a toolbox, which provides the realisation of cloud federation in a single cloud environments. In the toolbox, a multi-cluster infrastructure was deployed and the network characteristics over the links among the clusters were manipulated. In this way, different geographic distances, various network conditions and even network cut-off between each pair of clusters were emulated. With such a toolbox, one can easily establish a multi-cluster infrastructure in an OpenStack[1] environment and access "cloud federation in one box".

The aim of this toolbox is to provide an "easy-to-access", "easy-to-cooporate" and open source emulation platform, as leverage for developing control or networking solutions for cloud federation researches in the field of edge/fog computing. Tools like Prometheus[2] and Grafana [3] are also integrated in the toolbox for resource monitoring and experimental data tracing. We also deployed a service mesh layer in the toolbox with Istio[4] to enable communication for microservices deployed across multiple clusters.

My contribution in this work is mainly at network configuration for inter- and intra-cluster communication. I also set up the service mesh layer to enable communications among microservices across different clusters. I'm also involved in the preliminary design for the system architecture of the toolbox.

[1] `https://www.openstack.org`
[2] `https://prometheus.io`
[3] `https://grafana.com`
[4] `https://istio.io`

4.2 Conclusions and Future Work

In this thesis, we have walked through the development of an industrial network system, which shifted a MAC layer service function from local deployment to remote deployment in the cloud. Additionally, we looked at the potential of building a flexible multi-cluster federation toolbox, in order to provide a readily accessible emulation environment for developing solutions and applications in the field of cloud federation and edge/fog computing.

The first three papers have addressed the challenges of adopting single-cell RAN and cloud computing into industrial automation network system. Paper I has investigated the feasibility to accommodate heterogeneous IIoTs with the single-cell network. Paper II studied the viability of integrating the cloud in this RAN system, considering the inevitable delays introduced by having a split architecture and the cloud execution environment. In Paper III, we addressed the performance impacts of the cloud integrated system that was observed in Paper II, and proposed a scheduling strategy to mitigate this impact. In Paper IV, we built up a cloud federation toolbox and attempted to look at the microservice-based applications deployed across a broader geographically distributed system, taking into account inter-cluster network challenges.

The solutions in Paper III is proposed for a radio resource scheduling function, but not limited to it. The solution can also be adopted to a general scheduling process that runs into stochastic delay-constraint issue. Practically, in the case of load balancing or admission control of cloud computing, we would have more complex arrival processes for the inbound traffic. It is non-trivial to further investigate the compatibility of this system for more bursty and hybrid traffic than the ones described in the included papers, and further strengthen the solution for more realistic scenarios, with higher level of complexity. Furthermore, a cloud-native test-bed will be implemented to provided a view on the latency challenges practically and for testing developed solutions in a long term.

References

[1] Simson Garfinkel. *Architects of the information society : thirty-five years of the laboratory for computer science at MIT.* MIT Press, 1999. ISBN 0262571315. URL http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat07147a&AN=lub.1212761&site=eds-live&scope=site.

[2] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Special Publication 800-145, The US National Institute of Standards and Technology, 2011. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Last accessed Nov. 2020.

[3] 3GPP. System architecture for the 5G System (5GS); Stage 2 (Release 16); Version 16.6.0. Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), Spet. 2020. URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2440. Last accessed Nov. 2020.

[4] 5GPPP. Vision on Software Networks and 5G. White Paper final version 2.0, 5GPPP Software Network Working Group, Jan 2017. URL https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP_SoftNets_WG_whitepaper_v20.pdf. Last accessed Nov. 2020.

[5] 5GPPP. From Webscale to Telco, the Cloud Native Journey. Cloud Native White Paper Version: 1.0, 5GPPP Software Network Working Group, July 2018. URL https://5g-ppp.eu/wp-content/uploads/2018/07/5GPPP-Software-Network-WG-White-Paper-23052018-V5.pdf.

[6] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile Edge Computing - A key technology towards 5G. White Paper No. 11, European Telecommunications Standards Institute (ETSI), Sept. 2015. URL https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf. Last accessed Nov. 2020.

[7] Hannover messe events. URL https://www.hannovermesse.de/en/. Last accessed Nov. 2020.

[8] Prasad Satyavolu, Badrinath Setlur, Prasanth Thomas, and Ganesh Iyer. Designing for Manufacturing's 'Internet of Things'. White Paper, Cognizant, Jun 2014. URL https://www.cognizant.com/whitepapers/

Designing-for-Manufacturings-Internet-of-Things.pdf. Last accessed Nov. 2020.

[9] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (IIoT): An analysis framework. *Computers in Industry*, 101:1–12, Oct. 2018. ISSN 01663615. doi: 10.1016/j.compind.2018.04.015.

[10] ITU-R. IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. Technical Report ITU-R M.2083-0, Radiocommunication Sector of International Telecommunications Union (ITU-R), 2015. URL https://www.itu.int/rec/R-REC-M.2083-0-201509-I/en. Last accessed Nov. 2020.

[11] ITU-R. Minimum requirements related to technical performance for IMT-2020 radio interface(s). Technical Report ITU-R M.2410-0, Radiocommunication Sector of International Telecommunications Union (ITU-R), November 2017. URL https://www.itu.int/pub/r-rep-m.2410-2017. Last accessed Nov. 2020.

[12] ATIS IoT Categorization Focus Group. IoT categorization : Exploring the need for standardizing additional network slices. Technical Report ATIS-I-0000075, Alliance for Telecommunications Industry Solutions (ATIS), 2019. URL https://access.atis.org/apps/group_public/download.php/51129/ATIS-I-0000075.pdf. Last accessed Nov. 2020.

[13] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, 11:17–27, Mar 2017. ISSN 1932-4529. doi: 10.1109/MIE.2017.2649104.

[14] Ali Zaidi, Anders Bränneby, Ala Nazari, Marie Hogan, and Christian Kuhlins. Cellular IoT in the 5G era. White Paper GFMC-20:000025, Ericsson, 2020.

[15] Bernd Holfeld, Dennis Wieruch, Thomas Wirth, Lars Thiele, Shehzad Ali Ashraf, Jorg Huschke, Ismet Aktas, and Junaid Ansari. Wireless Communication for Factory Automation: an opportunity for LTE and 5G systems. *IEEE Communications Magazine*, 54:36–43, Jun 2016. ISSN 0163-6804. doi: 10.1109/MCOM.2016.7497764.

[16] 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), pages 1319–1324. IEEE, Mar 2017. ISBN 978-1-5090-5320-9. doi: 10.1109/ICIT.2017.7915554.

[17] Erik G. Larsson, Ove Edfors, Fredrik Tufvesson, and Thomas L. Marzetta. Massive MIMO for next generation wireless systems. *IEEE Communications Magazine*, 52(2):186–195, Feb 2014. ISSN 1558-1896. doi: 10.1109/mcom.2014.6736761.

[18] Elisabeth De Carvalho, Emil Bjornson, Jesper H. Sorensen, Petar Popovski, and Erik G. Larsson. Random Access Protocols for Massive MIMO. *IEEE Communications Magazine*, 55:216–222, May 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600477CM.

[19] Thomas L. Marzetta, Erik G. Larsson, Hong Yang, and Hien Quoc Ngo. *Fundamentals of Massive MIMO*. Cambridge University Press, 2016. doi: 10.1017/CBO9781316799895.

[20] Emil Bjornson, Elisabeth de Carvalho, Erik G. Larsson, and Petar Popovski. Random access protocol for massive MIMO: Strongest-user collision resolution (SUCR). In *2016 IEEE International Conference on Communications (ICC)*. IEEE, May 2016. ISBN 978-1-4799-6664-6. doi: 10.1109/ICC.2016.7510793.

[21] Emma Fitzgerald, Michal Pioro, and Fredrik Tufvesson. Massive MIMO Optimization With Compatible Sets. *IEEE Transactions on Wireless Communications*, 18(5):2794–2812, may 2019. ISSN 1536-1276. doi: 10.1109/TWC.2019.2908362.

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

[23] Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Maria Fazia, Massimo Villari, and Rajiv Ranjan. Internet of Things and Edge Cloud Computing Roadmap for Manufacturing. *IEEE Cloud Computing*, 3(4):66–73, Jul 2016. ISSN 2325-6095. doi: 10.1109/MCC.2016.91.

[24] Jakub Pizoń and Jerzy Lipski. Perspectives for Fog Computing in Manufacturing. *Applied Computer Science*, 12(3):37–46, 2016. URL `http://acs.pollub.pl/pdf/v12n3/4.pdf`.

[25] Cisco. Leading Tools Manufacturer Transforms Operations with IoT, 2019. URL `http://www.cisco.com/c/dam/en_us/solutions/industries/docs/manufacturing/c36-732293-00-stanley-cs.pdf`. Last accessed Nov. 2020.

[26] Baotong Chen, Jiafu Wan, Lei Shu, Peng Li, Mithun Mukherjee, and Boxing Yin. Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges. *IEEE Access*, 6:6505–6519, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2783682.

[27] Hesham El-Sayed, Sharmi Sankar, Mukesh Prasad, Deepak Puthal, Akshansh Gupta, Manoranjan Mohanty, and Chin Teng Lin. Edge of Things: The Big Picture on the Integration of Edge, IoT and the Cloud in a Distributed Computing Environment. *IEEE Access*, 6:1706–1717, 2017. ISSN 21693536. doi: 10.1109/ACCESS.2017.2780087.

[28] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, 5(2):1275–1284, apr 2018. ISSN 2327-4662. doi: 10.1109/JIOT.2018.2805263.

[29] Yehan Ma, Chenyang Lu, Bruno Sinopoli, and Shen Zeng. Exploring Edge Computing for Multi-Tier Industrial Control. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11), 2020. ISSN 0278-0070. doi: 10.1109/TCAD.2020.3012648.

[30] Paul Miller and Sanjay Bhatia. NFV, Cloud and SDN Moving Beyond Simple Virtualization. White paper, GENBAND and Intel Network Solution. URL `https://networkbuilders.intel.com/docs/Genband_nfv_whitepaper.pdf`. Last accessed Nov. 2020.

[31] China Mobile. C-RAN: The Road towards Green RAN. China Mobile White Paper ver 2.5, 2011. Last accessed Nov. 2020.

[32] ETSI. Cloud RAN and MEC: A Perfect Pairing. White Paper No. 23, European Telecommunications Standards Institute (ETSI), Feb 2018. Last accessed Nov. 2020.

[33] Jesper H. Sorensen, Elisabeth de Carvalho, and Petar Popovski. Massive MIMO for crowd scenarios: A solution based on random access. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 352–357. IEEE, Dec 2014. ISBN 978-1-4799-7470-2. doi: 10.1109/GLOCOMW.2014.7063456.

[34] 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. ISBN 978-1-4673-8894-8. doi: 10.1109/CloudTech.2016.7847722.

[35] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *2010 Proceedings IEEE INFOCOM*. IEEE, Mar 2010. ISBN 978-1-4244-5836-3. doi: 10.1109/INFCOM.2010.5461931.

[36] Ulrich Drepper. The cost of virtualization: Software developers need to be aware of the compromises they face when using virtualization technology. *Queue*, 6(1):28–35, January 2008. ISSN 1542-7730. doi: 10.1145/1348583.1348591.

[37] Lars Larsson, William Tärneberg, Cristian Klein, Erik Elmroth, and Maria Kihl. Impact of etcd deployment on kubernetes, istio, and application performance. *Software: Practice and Experience*, 50(10):1986–2007, 2020. doi: 10.1002/spe.2885.

Part II

Included Papers

Paper I

5G Radio Access Network Slicing in Massive MIMO Systems for Industrial Applications

A key enabler for Industry 4.0 is Fifth Generation Wireless Specifications (5G), within which network slicing is a promising technique to ensure customized quality of service for specific end-user groups in industrial scenarios. Massive Multiple Input Multiple Output (MIMO) plays a significant role in 5G but network slicing for massive MIMO has not yet been addressed. In this paper, we propose a network slicing scheme for a 5G Radio Access Network (RAN) with massive MIMO technology. Our simulations show that it is feasible to provide guaranteed performance in terms of high reliability, low latency, and a large number of connections by deploying our proposed scheme at the Medium Access Control Layer (MAC) layer of a massive MIMO base station.

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 De!ned Systems (SDS)*, Paris, France, pp. 262-267, April 2020.

1 Introduction

Wireless communication for factory automation is receiving growing interest in the context of Industry 4.0. Factory automation requires networks to provide extremely high reliability, high capacity, large throughput, and low latency as well as security and safety compliance to meet the performance requirements of different applications [1]. Today, the most common networking solutions for factory automation systems are wired networks, such as controller area networks, Modbus and industrial Ethernet. However, wires inhibit the mobility of end devices, and installation, management and maintenance of cables come with significant costs. While wireless networks overcome these shortcomings, most wireless standards, especially in unlicensed spectrum, struggle to meet critical requirements as cable networks can. As such, the design of industrial wireless standards is a vibrant research area.

5G, in combination with network slicing, brings a novel alternative solution that overcomes the flaws of most wireless networks[2]. The use cases that 5G covers meet many performance requirements for industrial automation, such as ultra-reliability and low latency [3], while also providing higher mobility, flexibility, and elasticity in the manufacturing system. A Global Mobile Suppliers Association (GSA) white paper [4] argues that network slicing will play a significant role in addressing the performance requirements in vertical industries. As defined in [5], network slicing permits tailoring of the network on the same physical infrastructure according to specific performance requirements from customers. A network slice allows isolated access to only the customers who are subscribed to it. Network slicing covers the industrial communication demand of channel separation even when using the same infrastructure. Studies on network slicing are very diverse, depending on the networking or communication technologies that they focus on, as well as the applications served by the slices. Network slicing is generally implemented by configuring a set of Virtual Network Function (VNF) and connecting them via virtual networks on a programmable infrastructure, such as is done in [6] and [7].

Some papers have proposed network slicing schemes in the RAN. However these solutions are highly dependent on the chosen RAN architecture. For instance, in [8], a framework is introduced to enforce network slicing in the RAN. The authors implemented a slice resource manager and resource mappers in the framework to perform a two-level scheduling process. [9] presents a radio resource slicing framework based on LTE-A air interfaces, while [10] proposes a slicing plane for RAN considering inter-cell interference and resource grid isolation. To the best of our knowledge, there are no RAN-based network slicing schemes proposed for 5G networks that use massive MIMO technology. Massively many connections and low latency communications are from two

different traffic categories that massive MIMO can serve, but few studies have treated these two types of traffic separately and accommodated them on the same infrastructure.

In this paper, we propose a RAN-based network slicing scheme for 5G networks using massive MIMO technology. Our slicing scheme uses a coordinated two-level scheduler and specifically targets industrial scenarios, aiming to provide Quality of Service (QoS) guarantees on par with wires for factory automation. Although a two-level scheduling mechanism was exploited in some slicing solutions such as [8], we address a different type of radio resource abstraction based on massive MIMO and focus on industry use cases. Our results demonstrate that our scheme facilitates wireless communication for factory automation, meeting the QoS requirements of low latency and high reliability and enabling massively many connections for the specific scenarios we investigate. We also guarantee radio resource isolation between the slices.

2 Targeted System

In this section, we describe the targeted system that we use to evaluate our proposed network slicing scheme. We focus on a factory automation scenario. An industrial facility has numerous sensors, controllers and actuators, here called industrial units, which are all connected via a 5G network using massive MIMO antenna technology. We assume that the distance between the industrial units is small enough that they can all be covered by the same Base Station (BS); that is we have a single cell environment. The industrial units have different priorities, for instance the control units should have higher priority since their application is more time critical. Thus, we categorise the industrial units and subscribe them to two different network slices on top of our single cell system.

We take as our system setting massive MIMO with Time Division Duplex (TDD) and Orthogonal Frequency-Division Multiplexing (OFDM) modulation [11]. The time-frequency space of a single massive MIMO system can be divided into so-called coherence blocks, where a coherence block is the largest time period during which the channel can be viewed as time-invariant and the channel frequency response is approximately constant. A coherence block is shared by uplink data, downlink data and uplink pilot transmissions. The uplink pilots are used by the BS to estimate each end-user's Channel State Information (CSI), used by the BS for precoding needed to process the input and output data. Thus a pilot is needed for a given end-user to transmit data successfully and we will consider the uplink pilots as the resources that the end-users require before a transmission can start.

Each coherence block can host at most p mutually orthogonal pilot signals, and pilots with the same length yield the same performance. Ideally, the maximum number of end-users a system can serve is $K = p$, giving equal performance to the users. However, in practice, some end-users may be assigned multiple pilots, giving a more accurate CSI estimation. Hence the number of users that can be served can be less than p.

In this paper, we assume that each industrial unit in our factory automation scenario uses either the Ultra-Reliable and Low-Latency Communication (URLLC) or Massive Machine Type Communication (mMTC) traffic types. The actuation units and controllers are distributed and their feedback loops are connected via the mobile network. Sensors are also deployed to measure the states of the actuators. Communications for the control loops are all categorised as URLLC type traffic. Meanwhile, there are also numerous devices sending out occasional monitoring messages about the plant. These monitoring devices are installed and distributed throughout the whole plant and require massively many connections. They are thus categorised as mMTC.

We separate the industrial units into two network slices with different performance requirements and priorities. We define two network slices: 1) a URLLC slice, denoted by S_1, in which the end-users send periodic transmission requests; and 2) an mMTC slice S_2, serving a large number of end-users with aperiodic transmissions. We assume that the total number of connected industrial units exceeds the number of antennas of the massive MIMO system. It is, therefore, impossible to provide a sufficient number of resources (pilots) for each individual unit in each coherence block. Instead, a dynamic scheme is needed in order to meet the QoS requirements.

3 Proposed slicing method

In this section, we present our RAN-based network slicing method. The proposed method: 1) enables the coexistence of two traffic classes with different QoS requirements; 2) isolates part of the radio spectrum resources for one high priority slice in order to meet a certain network performance; 3) prevents the end-users being interfered with the users of the second, low-priority, slice; and 4) can be generalized to more than two slices, facilitating a network slicing solution for several traffic classes with different priorities and QoS requirements.

We assume that during one slot time T_{slot}, the channel is time-invariant; hence a slot time is equal to one coherence interval. A device may transmit on both the uplink and downlink if it is granted a pilot signal within a coherence block. We also assume that the resources in the coherence block are sufficient to complete a full packet transmission from each end-user assigned a pilot by

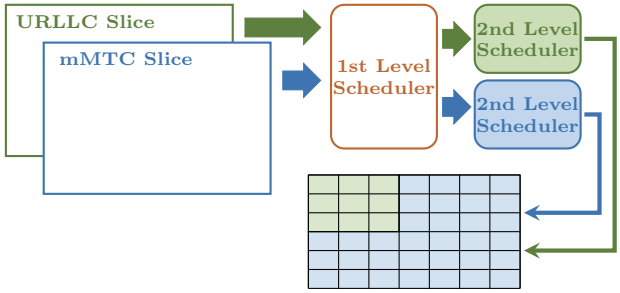


Figure 1: An example of the two-level MAC scheduler tailoring the time-frequency resource elements of a coherence block for two network slices. The first level scheduler allocate the resources to each slice and each second level scheduler assigns the pilots to the subscribed end-users.

the scheduler.

The number of customers served by each slice is K_1 and K_2, respectively, giving that $K = K_1 + K_2$. We consider that each user in S_1 acquires p_1 pilot signals and p_1 may be larger than one in order for the BS to obtain more accurate CSI and provide higher channel reliability. In S_2 we assume that $p_2 = 1$, to serve as many devices as possible for mMTC. During each coherence interval, we consider that there are $\tilde{K}_1$ and $\tilde{K}_2$ end-users with transmission requests. To serve all the devices, the following condition should be satisfied.

$$\tilde{K}_1 p_1 + \tilde{K}_2 p_2 \leq p. \tag{1}$$

Our proposed network slicing method is designed as a two-level MAC scheduler, as illustrated in Figure 1. The first level of the MAC scheduler is responsible for inter-slice scheduling. It divides the radio spectrum into two and maps dedicated resources to each slice. The second level enforces intra-slice scheduling and allocates the pilots assigned by the first level to the devices within the slice. The end-users interface only with the slices they are subscribed to and receive pilots from the second level schedulers.

The first level scheduler conducts a simple priority scheduling wherein slice S_1 always takes precedence over slice S_2 to guarantee its QoS requirements. The scheduler always assigns the required number of pilots to slice S_1 until all pilots are in use. If there are any pilots unassigned after this, they will be assigned to slice S_2. Each second level scheduler will then use a scheduling algorithm for intra-slice resource allocation to individual users. In this paper, we will investigate three commonly deployed scheduling algorithms.

First Come First Served (FCFS): The scheduler maintains two queues

for all the incoming requests from the two slices and serves them in order of arrival time. The advantage of FCFS is that the reliability of S_1 will always be guaranteed when the traffic does not exceed the highest load that the system can tolerate, and there will not be any resource waste. However, it requires the scheduler to be aware of the full queuing information and would cost extra computation and signalling to maintain and acquire this information.

Round Robin with partial queuing information (RR_Q): In this method, the scheduler does not need to acquire full queuing information. During each coherence interval, it instead iterates through all the devices in each slice and assigns pilots to whichever users are signalling for transmission. In this way the overhead would be lower than FCFS since RR_Q does not assign plots in order of arrival time.

Round Robin without queuing information (RR_NQ): In this method the scheduler does not acquire any queuing information; instead it assigns pilots to each device subscribed to the slice, whether it signals for transmission or not. This yields a static scheduling approach. The scheduler groups a number of slots into a frame for pilot allocation such that a frame time is equal to the deadline of each request in S_1. In this way, we guarantee that every request will be assigned a pilot within its deadline. During a frame, the scheduler iterates through all the devices in S_1 assigning pilots. If there are pilots left during a frame, they will be allocated to S_2. This does not incur any overhead for searching for active devices. However it will result in a large number of wasted pilots and is not applicable to S_2, since S_2 hosts an enormous number of end-users. Accordingly, the other methods are utilized within S_2 when RR_NQ is applied within S_1.

The second level scheduler of each slice can choose a different scheduling algorithm. We will show in the following sections that the chosen scheduling algorithm of one slice has very little impact on the QoS in the other slice.

4 Simulation

We evaluated our proposed network slicing method using simulations. The complete code of our simulation program, experiments and data will be shared on Github [12] upon publication. The implementation is based on a massive MIMO MAC layer simulator[13]. The simulation program is written in Python and is extensible with any applicable scheduling algorithm on both levels of the scheduler. It also supports custom traffic profiles on each slice by specifying the inter-arrival distribution, deadline, required number of pilots and number of end-users.

Table 1: Network parameters and variables used in the simulation.

Variable name	Value	Symbol
Simulation length	10000 ms	L
Slot time	0.5 ms	T_{slot}
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, RR_Q or RR_NQ	

4.1 Simulation model

In our simulation, a predefined number of end-users (representing industrial units) are subscribed to each of the two slices described in Section 2. The end-users belonging to a slice have a specific arrival distribution representing the assumed traffic profile for these types of industrial units. The scheduler then assigns pilots to end-users as described in Section 3.

Each end-user in the URLLC slice S_1 sends transmission requests with a near-constant period of value d_1 times a small, normally distributed variance $\omega_1 \sim \mathcal{N}(1, 0.05)$. The other slice S_2 however follows a Poisson distribution with a larger average inter-arrival time d_2, representing aperiodic transmission requests from monitoring sensors. The deadlines of the transmission requests are the same as their average inter-arrival times. If a pilot is granted before the deadline is reached, the connection between the device and the BS is initiated and data transmission succeeds within the corresponding coherence time. If a deadline is missed, the packet is dropped.

The massive MIMO MAC layer simulator provided in [13] is an event-based simulator with basic traffic generation functions and scheduling implementations. We extended it so that the end-users are separated into different slices according to different traffic profiles and the pilots are allocated via our two-level scheduling process. Each transmission request is sent when a request event is triggered according to the aforementioned traffic profiles of the end-users. A request is labeled with its slice type, arrival time (the same as the time when the request event is triggered) and deadline (the time when the request expires if no pilot is assigned to it).

Every T_{slot} a new scheduling event is triggered, in which p pilots are released to be assigned to the requests. Each scheduling event handles all the active requests. It marks a request as expired if it has passed its deadline without receiving a pilot. Afterwards it assigns pilots to the requests with later deadlines using the two-level scheduler, until all pilots are used for the current slot time. A request that gets sufficient pilots for its CSI transmission is marked as served. In slice S_2, each end-user takes only one pilot in our simulation ($p_2 = 1$), obtaining the least sufficient CSI quality for the transmission. However in slice S_1, which requires higher reliability, each end-user may need more than one pilot to guarantee accurate CSI, yielding $p_1 \geq 1$, depending on the demands of the users. All unhandled request events are marked as pending and are still active until the next scheduling event.

4.2 Experiments

In our experiments, we tested different combinations of the scheduling algorithms described in Section 3 to investigate how an increase of traffic load in each slice affects the QoS performance. We define the traffic load ρ_i in slice S_i in terms of the number of users K_i that is subscribed to it.

$$\rho_i = \frac{K_i p_i T_{slot}}{p d_i}, \ i = 1, 2. \tag{2}$$

The network parameters we used in the simulation are shown in Table 1. These parameters are based on those of our 100-antenna test-bed [14] under a high mobility scenario. Additionally, we evaluated our method according to three different aspects: the performance of the URLLC slice, the performance of the mMTC slice and the isolation between the two slices. In the following we describe the experimental setup for the three evaluations.

Latency and reliability performance in URLLC slice

To evaluate the latency and reliability performance versus the growth of traffic load ρ_1 in slice S_1, we kept the traffic load ρ_2 in S_2 a constant value 0.5 and applied FCFS as the second level scheduler for S_2. In each simulation, ρ_1 is generated via the profile shown in Table 2. In this experiment the end-users in S_1 required a short latency and a moderate channel reliability. All three scheduling algorithms were applied in S_1 for the evaluation.

Isolation between the two slices

In this experiment, we investigated whether the performance of slice S_2 will be affected by the selection of the second level scheduler in the higher priority

Table 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	–	–	RR_NQ
S_2	ρ_2	0.5	–	–
	p_2	1	1	1
	d_2	50 ms	50 ms	50 ms
	scheduler	FCFS	RR_Q	–

slice S_1. Therefore in this case we kept the traffic load ρ_1 in slice S_1 a constant value 0.5 and applied the parameters in Table 2 to generate the traffic load in S_2.

Latency and connection performance in mMTC slice

In this experiment, we evaluated the QoS performances in slice S_2. We investigated how the latency performs with the growth of the traffic load ρ_2 in S_2. Thus we also had a constant traffic load $\rho_1 = 0.5$ in slice S_1. In this experiment, we applied the RR_NQ scheduler in S_1 and evaluated both the FCFS and RR_Q methods in slice S_2. The traffic load ρ_2 was generated by following the parameters in Table 2. In this experiment, we also investigated the limit of the number of end-users that the system can accommodate in S_2 with our parameters.

5 Results and discussion

In this section, we present and discuss the results of our evaluation. We focus on the average waiting time and the loss rate of the requests to each slice when applying our network slicing scheme. We note that the 95% confidence intervals of the average waiting times from our experiments are all within 1.6% of the mean.

5.1 Latency and reliability performance in URLLC slice

First, we discuss how different scheduling algorithms perform versus the increasing traffic load ρ_1. Since S_1 has a higher priority there will not be any impact from the traffic load of S_2 on it. Thus, there should not be any packet

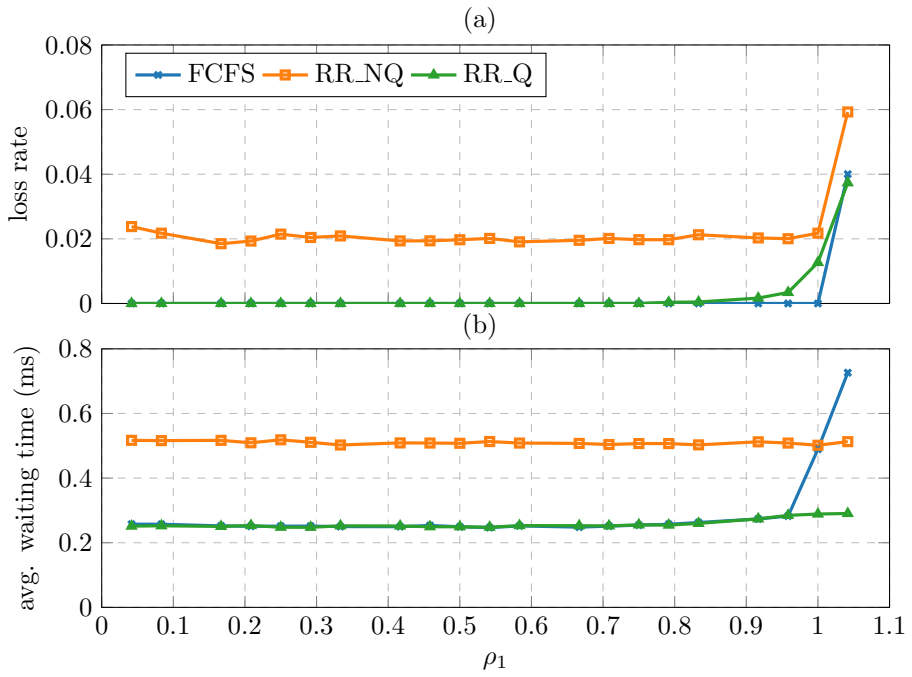


Figure 2: Slice S_1 (a) loss rate and (b) waiting time with three different schedulers. The RR_NQ method has around 2% packet loss even when the system is lightly loaded. The RR_NQ method has a higher latency compared to the other schedulers. When using FCFS,the average waiting time of the requests grows rapidly once the system is fully loaded.

loss when ρ_1 is less than 1.0. However as shown in Figure 2(a), the RR_NQ scheduler does have an impact on the loss rate even when the traffic load is small. This loss is due to the variance of the inter-arrival time of the requests from S_1, considering a case where two subsequent requests from the same device have an inter-arrival time slightly less than d_1 due to the introduced variance. The second request would possibly fail to be allocated a pilot if the first one just arrived at or shortly after the allocation slot in the previous frame. Thus this loss rate is independent of the arrival period and depends only on the arrival rate variance. It is also possible to have such an impact of loss rate on the RR_Q method, especially when the deadline is short (e.g. 1 ms) and the variance on the inter-arrival time is relatively large, as we can see from

Figure 2(a) that RR_Q has an increase in loss rate when S_1 is heavily loaded. This variance is likewise reflected in the variance of loss rate when using the RR_NQ scheduler.

It is also shown in Figure 2(b) that the RR_NQ scheduler has an effect on the average waiting time of the requests. The average waiting time is equal to half a slot time for the FCFS and RR_Q schedulers since the BS iterates through all the arrival requests every slot time. However with RR_NQ the average waiting time is half a frame time because the BS iterates through all the devices each frame. Furthermore, when ρ_1 reaches 1.0, the waiting time of the FCFS scheduler increases rapidly and almost reaches the deadline of each request on that slice when $\rho_1 = 1.1$. This is because the waiting time with FCFS depends on the queues of all the arrival requests as maintained by the BS, but in the other two cases, it depends on the length of each queue on the device side. Note that this waiting time is calculated from when a packet arrives to the moment when it is assigned a pilot.

There is a trade-off between the computation and signalling cost, and the performance of the different schedulers. The RR_NQ saves on signalling however it is not applicable to critical applications that require ultra-reliability. FCFS gives a good performance, especially when the system is not fully loaded, but it costs heavily in computation and signalling. When the system is almost fully loaded, FCFS may not guarantee reliability as bursty traffic can quickly fill up the arrival queues.

5.2 Isolation between the two slices

The effect on the QoS performance of S_2 of different schedulers in S_1 reflects the isolation performance of our slicing scheme. Figure 3 shows a slight difference with the RR_NQ scheduler than with the other two. Under the same traffic load, the average waiting time in S_2 also depends on the traffic profile in S_1.

As shown in Eq. (2), the number of end-users K_1 can be larger when d_1 is longer for the same traffic load. However, with K_1 customers, it takes the base station $K_1 p_1/p$ slots to serve all the devices in S_1, which is longer than it would take if the traffic profile had a shorter deadline, and this would slightly increase the average waiting time of S_2. For the same reason, this difference in waiting time will be smaller when the deadline is shorter since the waiting time is bounded to the frame time when using the RR_NQ scheduler. This means that isolation is not guaranteed by the scheduler since the performance of the waiting time in S_2 depends on the deadlines of the traffic in S_2.

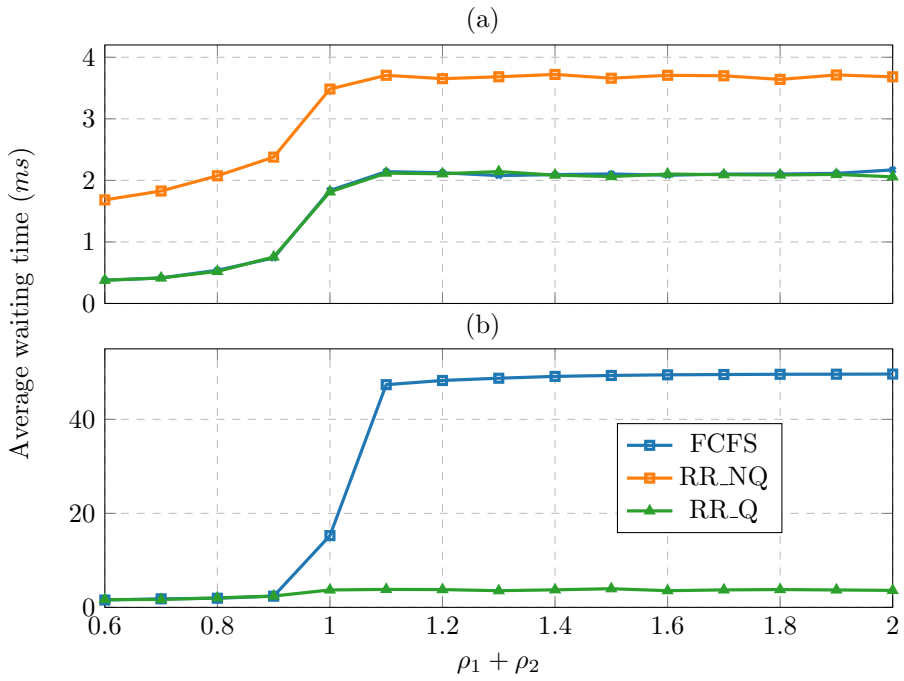


Figure 3: Slice S_2 average waiting time (a) when applying three different schedulers in S_1 along with RR_Q in S_2 and traffic parameters in Eval. (2) of Table 2, among which scheduler RR_NQ results in a higher waiting time than the other configurations; (b) when applying FCFS and RR_Q in S_2 along with RR_NQ in S_1. The x axis shows the overall system traffic load $\rho_1 + \rho_2$ where in this case ρ_1 is constant at 0.5. All the requests have the deadline of 10ms in this experiment.

5.3 Latency and connection performance in mMTC slice

To investigate the impact of different scheduling algorithms used in S_2, we plotted the waiting time of S_2 as ρ_2 increases. In this case we have a constant traffic load of 0.5 in S_1, which causes a traffic block in S_2 starting from $\rho_2 = 0.5$, as shown in Figure 4. The 95% confidence intervals on loss rate are within 5% of the values shown in the figure. Figure 4 illustrates loss rate versus number of connections in S_2 with corresponding overall system traffic load at the top of the figure. Using our specified traffic profile, the system can host up to 600

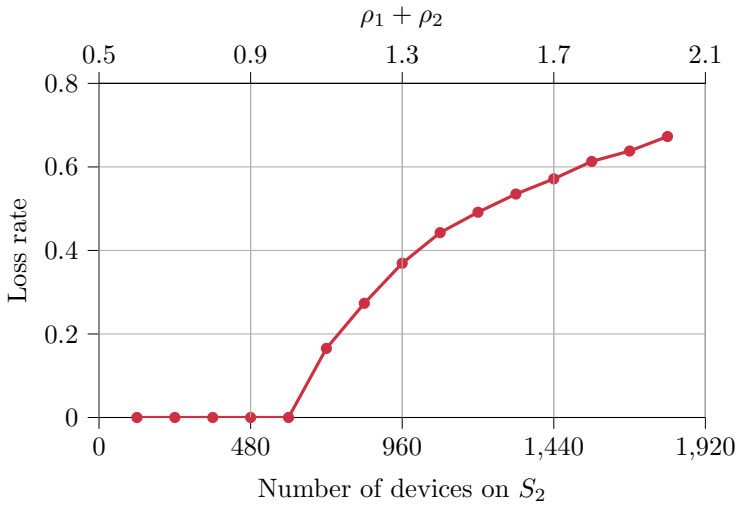


Figure 4: Slice S_2 loss rate versus number of connections. With the corresponding parameters, the system can host up to 600 end-users in S_2 without any loss. This is equivalent to an overall traffic load $\rho_1 + \rho_2 = 1$.

end-users simultaneously without any loss under both the FCFS and RR_Q schedulers. In S_2, there is no impact from the type of scheduler because RR_NQ is not applicable to this slice and the mean inter-arrival time is long compared to S_1.

The schedulers' performance in S_2 is however similar to S_1 as shown in Figure 3. There, the waiting time with the round robin method does not increase when the system is overloaded. This is because the system blocks devices exceeding those that can be accommodated in the round robin. The waiting time performance is thus the same as when the traffic load $\rho_1 + \rho_2 = 1$ since we only calculate waiting time for those requests that were assigned pilots.

The simulation results show that the performance of S_2 is affected by the traffic loads in both slices and by the choice of its own scheduler, but not by the choice of scheduler in S_1. Since S_2 does not require ultra-reliable performance, we may allow some packet loss in order for the system to host a larger number of connections to the network in S_2.

6 Conclusion

In this paper we have presented a resource slicing scheme for a single-cell RAN based on massive MIMO. We proposed a two-level MAC scheduler that divides the radio resources into two parts and maps them to each slice, and is also responsible for intra-slice resource allocation. We considered three different scheduling algorithms (FCFS, RR_Q and RR_NQ) that could be applied in our slicing scheme. Simulation experiments were performed to evaluate the system performance in accordance with the user requirements of each slice.

Since massive MIMO systems have all the signal processing and channel estimation only on the BS side, it may cause lower power efficiency if the computation on the MAC scheduling consumes a lot. RR_NQ scheduling has the lowest overhead since it is not necessary for the BS to maintain a queue of all incoming requests or for the end-users to signal for transmission requests. However this method does not guarantee ultra-reliability since there can be packet loss due to the statistical scheduling approach. Further, it is not applicable for the case of massive IoT connections since the number of devices is far more than the available pilot resources. FCFS scheduling is more reliable and applicable but may have much higher overhead on the BS for maintaining the arrival queues. Meanwhile the RR_Q method has a large signalling overhead.

Our slicing scheme can be extended to host more than two slices or apply other scheduling algorithms to meet diverse performance requirements. The next step of our work is to extend the proposed scheme to a complete network slicing approach that combines RAN slicing with Core Network slicing in an optimal way.

Acknowledgement

This work is partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Maria Kihl and Emma Fitzgerald are also partially supported by the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the 5G-PERFECTA Celtic Next project funded by Sweden's Innovation Agency (VINNOVA). Maria Kihl and Emma Fitzgerald are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT) strategic area.

References

[1] Giuliana Zennaro, Aleksandra Stojanovic, and Marina Giordanino. Report on vertical requirements and use cases. Technical Report 761536, 5G-TRANSFORMER Project, 2017. URL `http://5g-transformer.eu/index.php/deliverables/`. Accessed: Mar, 4, 2020.

[2] M. Gidlund, T. Lennvall, and J. Åkerberg. Will 5g become yet another wireless technology for industrial automation? In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 1319–1324, March 2017. doi: 10.1109/ICIT.2017.7915554.

[3] 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. Technical report, 5Gppp, April 2016. URL `http://www.5g-ppp.eu/`. Accessed: Oct. 23, 2019.

[4] GSA. 5g network slicing for vertical industries. White paper, Global mobile Suppliers Association, September 2017. URL `https://www.huawei.com/minisite/5g/img/gsa-5g-network-slicing-for-vertical-industries.pdf`. Accessed: Oct. 23, 2019.

[5] GSMA. An introduction to network slicing. Technical report, GSM Association, 2017. URL `\url{https://www.gsma.com/futurenetworks/resources/an-introduction-to-network-slicing-2/}`. Accessed: Oct. 23, 2019.

[6] 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. *IEEE Communications Magazine*, 55(5):80–87, May 2017. ISSN 1558-1896. doi: 10.1109/MCOM.2017.1600935.

[7] 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)*, pages 1–6, Dec 2018. doi: 10.1109/GLOCOMW.2018.8644145.

[8] Adlen Ksentini and Navid Nikaein. Toward enforcing network slicing on RAN: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017. ISSN 01636804. doi: 10.1109/MCOM.2017.1601119.

[9] Adnan Aijaz. A radio resource slicing framework for 5g networks with haptic communications. *IEEE Systems Journal*, 12(3):2285–2296, 2017. ISSN 1932-8184. doi: 10.1109/jsyst.2017.2647970.

[10] 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, page 237–238, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329897. doi: 10.1145/2620728.2620782.

[11] Thomas L. Marzetta, Erik G. Larsson, Hong Yang, and Hien Quoc Ngo. *Fundamentals of Massive MIMO*. Cambridge University Press, 2016. doi: 10.1017/CBO9781316799895.

[12] Massive MIMO slicing. URL `https://github.com/HaoruiPeng/slicing-simulator`. Accessed: Oct. 23, 2019.

[13] Massive MIMO simulator. URL `https://github.com/jost95/massive-mimo-simulator`. Accessed: Jun. 28, 2019.

[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)*, pages 287–293, Dec 2014. doi: 10.1109/GLOCOMW.2014.7063446.

Paper II

Massive MIMO Pilot Scheduling over Cloud RAN for Industry 4.0

Cloud-RAN (C-RAN) is a promising paradigm for the next generation radio access network infrastructure, which offers centralized and coordinated base-band signal processing in a BBU pool. This requires extremely low latency fronthaul links to achieve real-time signal processing. In this paper, we investigate massive MIMO pilot scheduling in a C-RAN infrastructure under a factory automation scenario. We use simulations to provide insights on the feasibility of C-RAN deployment for industrial communication, which has stringent criteria to meet Industry 4.0 standards. Our experiment results show that, concerning a pilot scheduling problem, the C-RAN system is capable of meeting the industrial criteria when there is fronthaul latency in the order of milliseconds.

Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl, "Massive MIMO Pilot Scheduling over Cloud RAN for Industry 4.0", *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Hvar, Croatia, 2020, pp. 1-6.

1 Introduction

In the context of Industry 4.0 and Internet of Things (IoT), the communication networks are expected to evolve towards wireless communication, which should have characteristics of high reliability, high capacity, large throughput and low latency to meet the performance requirements of different industrial applications. The Fifth Generation Wireless Specifications (5G) promises to provide these characteristics and thus is envisioned to be one of the future infrastructures for industrial communication networks. Cloud-RAN (C-RAN) is an intriguing candidate Radio Access Network (RAN) architecture for 5G that enables softwarization and resource centralization in radio access networks and promises to provide mobile Internet access with low cost and highly efficient network operations.

The basic concept of C-RAN is to detach the Base-Band processing Unit (BBU) from multiple legacy radio base stations and centralize them into a BBU pool. The remaining RRH are only equipped with basic radio-frequency functionalities like transmitting, receiving and analog/digital conversion. The BBU pool allows for base-band signal processing in a cooperative way for multiple Remote Radio Head (RRH) sites.

However, as of now various challenges remain to be solved in order to deploy the C-RAN infrastructure for the next generation mobile networks as explained in [1, 2]. One important challenge is to establish the fronthaul links that enable communication between the BBU pool and RRHs. These fronthaul links must comply with the stringent bandwidth and latency requirements for C-RAN.

Massive Multiple Input Multiple Output (MIMO) is another essential enabler for the next generation RAN that significantly increases the system capacity in order to handle the rapid growth of traffic in mobile networks. However, these large scale antenna systems require a huge amount of computational power for base-band signal processing. Therefore, it would be beneficial to adopt massive MIMO in C-RAN and to split part of the processing functionalities to a remote BBU pool. However, offloading the computational resources of such large antenna systems to a remote BBU pool implies that they may suffer from latency limitations while transmitting enormous amount of data to the computational unit.[3].

C-RAN systems build on cloud-native technologies, which also leads to problems for real-time processing, since the virtualization technology introduces more layers on the data path along the processing chain. These characteristics of the C-RAN system cause long-tailed delay and jitter in the RRH-BBU communication. This could also introduce catastrophic interruptions in the real-time signal processing [4].

In order to deploy C-RAN infrastructure with massive MIMO for industrial

automation networks and meet the stringent performance requirements, we must show that the latency in the system does not collapse the network function performance and that the impact of the delay can be mitigated with simple strategies. This system is only viable if the massive MIMO signal processing chain can guarantee that the performance in terms of communication reliability and connectivity still meet the industrial criteria when the data transmission between two functions are delayed.

As both massive MIMO and C-RAN are the most competitive candidates for building up the infrastructure of future mobile radio access networks, investigations on the combination of the two techniques have received a lot of interest. In [5, 6], the functionality split in massive MIMO RRH C-RAN system is addressed to tackle the bandwidth fronthaul limitation. Instead of offloading the whole base-band function chain to the BBU, the authors keep part of the function blocks in the RRH and allow them to be processed locally.Other solutions to the limited-fronthaul in massive MIMO C-RAN system are investigated as well. A prefiltering C-RAN architecture is proposed in [7] to compress the link data rate over the fronthaul and to keep the RRH structure as thin as possible. In [8], pilot contamination and imperfect channel estimation are considered as the impacts of the limited fronthaul. In [9], the authors proposed a decision-theoretic framework to tackle the delayed Channel State Information (CSI) for a rate allocation problem in C-RAN and optimize the end-to-end TCP throughput performance for the mobile edge cloud users. In their formulation, the TCP response latency experienced by the users is considered as a constraint and only a low mobility scenario is addressed.

To the best of our knowledge, with regard to the research on massive MIMO with C-RAN, the pilot scheduling problem has not yet been addressed. Likewise, few have considered the latency as the main constraint in the fronthaul in their problem formulation, however latency significantly affects both scheduling performance and user experience.

In this paper, we target a C-RAN system, which introduces delays to the single processing chain due to the latency constraint fronthaul and cloud environment. In that context, we address the pilot scheduling function at the Medium Access Control Layer (MAC) layer of massive MIMO that is implemented in the BBU pool of the addressed C-RAN system. We focus on the feasibility of deploying such a system under industrial automation requirements from the perspective of the scheduling performance, which is affected by several factors of the system. To address the challenge, we applied a commonly used Earlist Deadline First (EDF) strategy on the pilot scheduling problem to evaluate a latency constrained system using simulations. Our investigations show that C-RAN is capable of providing a reliable communication infrastructure that meets the criteria for industrial automation.

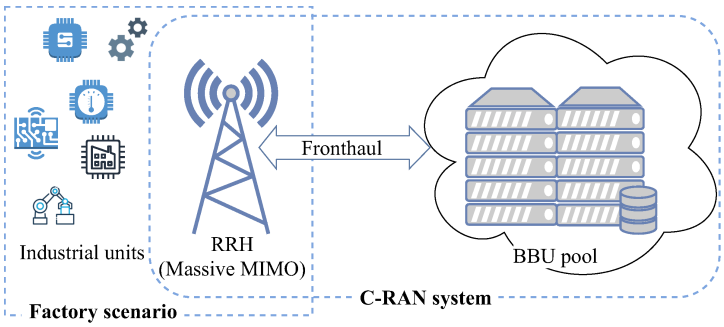


Figure 1: Target system architecture.

2 Targeted System

In this paper, we target a C-RAN architecture that includes one BBU pool and one massive MIMO RRH, connected with a fronthaul link, shown in Figure 1. As the MAC layer scheduling function is the main focus of our problem, we assume that the Physical Layer (PHY) functionalities are operated on the RRH and no raw base-band data blocks are transmitted over the fronthaul link. Thus we neglect the bandwidth limitation.

2.1 C-RAN System

In C-RAN, the traditional distributed base-band processing units (BBUs) are detached from the radio-frequency processing units (RRHs) and are centralized into a BBU pool. The remaining RRHs are co-located with the antenna while the BBU pool is responsible for the base-band processing of multiple RRHs. The BBU pool is connected to the target RRHs by fronthaul links. For a manufacturing process, the communication distance is normally less than 100m [10], thus we assume that all the units can be covered by the radio range of one RRH in our target scenario.

The fronthaul link between the RRH and the BBU pool could be up to 40km long[11]. Due to this geographic separation and the cloud environment in the BBU pool, the C-RAN architecture inevitability incurs delay between network functions and essentially breaks down the signal processing chain of an access network. The permissible round-trip delay of the fronthaul link varies from 5µs to 400µs depending on different techniques and function splits [12]. However, in Section 4 we show that the round-trip delay in our target system could be up to milliseconds, which imposes more interruptions in the processing chain.

2.2 Massive MIMO and Radio Resources

We use massive MIMO as the RRH of our target system. The time-frequency space of a single massive MIMO system can be divided into coherence blocks, which is the largest time interval during which the channel can be viewed as time-invariant and where channel frequency response is approximately constant for an end-user. A coherence block is shared by uplink data, downlink data and uplink pilot transmissions. The uplink pilots are used by the base station to estimate each end-user's CSI, which is needed for precoding to process the input and output data [13]. Thus, in every coherence interval, a new pilot is needed for a given end-user to transmit data successfully. In this paper, we consider the uplink pilots as the resources required by the end-users in an industrial automation scenario before a transmission can start.

The length of a coherence interval mostly depends on the end-users' mobility when the carrier frequency is fixed [13]. An end-user with lower moving velocity yields a longer interval, therefore it requires fewer pilots to transmit the same amount of data compared to one with higher mobility.

2.3 Industrial Communication Network

We address an indoor industrial automation scenario, where there are numerous sensors, controllers and actuators, here called Critical Units (CUs), which are part of a dynamic control system and are interconnected by a wireless industrial network. The traffic generated by the control operations with these units has key requirements such as less than 10ms latency, availability within the range of 95%-99.999% and density of 10000 devices per km^2, but the mobility of these units are mostly fixed or very low, since there is usually an indoor environment for industrial automation [14].

Because of the processes' low latency requirement, in this paper we assume that each transmission request has a hard deadline. If a unit has not been assigned a channel resource within the deadline, the transmission attempt failed and the data is discarded. Also, as most units have low mobility in the scenario, the coherence interval in the massive MIMO time-frequency space can be relatively long, and thus, a larger number of units can be served by the radio system simultaneously.

To complicate things, there are many types of units in such a system, some with less stringent requirements and thus, the priority of such units is lower than the CUs. The traffic generated by these units is considered as background traffic in the system. Therefore, it is important to optimize the radio resources allocated to the prioritized traffic from CUs, since the remaining resources can be allocated to the low-priority background traffic.

2.4 Pilots Scheduling Strategy

In our targeted industrial setting, the requests from CUs have strict deadlines but the number of pilots in a coherence interval is limited. In order that the CUs get assigned the pilots for data transmissions within their deadlines, we need to deploy a MAC scheduler to allocate the pilots. In our targeted C-RAN system, the scheduler is located in the BBU pool. The objective of the scheduler is to serve as many requests as possible within their deadlines. The background traffic will be served if there are pilots left in each coherence interval after the requests from CUs have been scheduled.

When allocating pilots to the CUs, the massive MIMO RRH follows the decisions made by the remote scheduler to allocate the pilots to the CUs. In order to investigate the feasibility of C-RAN deployment for industrial automation scenarios, we applied a scheduling strategy with EDF policy on the MAC layer to allocate the pilots to the CUs. The EDF policy guarantees that the CUs whose requests have earliest deadlines get the pilots first.

We propose the two following performance metrics for investigating how massive MIMO pilot scheduling is affected by the C-RAN constraints.
Loss (L): A request is dropped if it is not scheduled within its deadline. The loss can be calculated as the ratio between the dropped transmissions and the total number of requests.
Pilot utilization (U): A pilot is wasted every time it is allocated to a CU that has nothing to send. The utilization of pilots can be calculated as the ratio between the pilots that are successfully assigned for transmission requests and the total number of pilots that are allocated.

3 Simulation Model

In this section, we present the system simulation model shown in Figure 2, given that in total K active CUs are covered by the radio range of the RRH. The RRH communicates with the BBU pool via the fronthaul link in order to allocate pilots to the CUs.

3.1 RRH and BBU Pool Model

We consider that each CU only needs one pilot for the base station to estimate its channel state information in order to serve the transmission requests in a coherence interval. We assume that the number of available pilots in an interval is proportional to the length of the interval, which is determined by the mobility of the CUs. Since the end-users can be multiplexed in the spatial domain in massive MIMO, if a given CU gets assigned a pilot, we consider

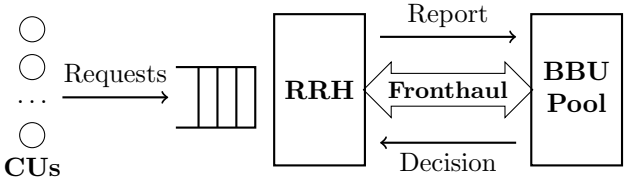


Figure 2: Simulation model.

that the number of its requests that can be served is also proportional to the interval length.

We denote the minimum interval length of our system as T_c, during which p pilots are available, implying that maximum p CUs can be assigned the pilots during T_c and one transmission request from each CU is served if it is assigned a pilot. We also denote by T_{slot} the actual length of a coherence interval, as well as as an allocation time slot in our scheduling problem, and there are P pilots available during each slot. When $T_{slot} = T_c$, we call it a high mobility scenario. When T_{slot} increases, it yields that the units in the scenario have lower moving velocity, and the number of available pilots P during T_{slot} increases proportionally.

The RRH keeps an ingress queue of all the active transmission requests. The BBU is able to keep track of the status of this queue. Every time the BBU gets updated queuing information, it sends a new scheduling decision so that the RRH could apply the updated allocation policy to the active CUs.

3.2 Traffic Model

Each CU_k, where $k \in \{1, 2, ..., K\}$, sends out the transmission requests at an average rate λ_k. We take into account the industry and IoT source level traffic models summarized in [10]. We use *Homogeneous periodic traffic* as the arrival process to generate transmission requests. By following this arrival process, each CU sends out requests with a nearly constant period around c but with a normally distributed noise, implying the average arrival rate of CU_k is $\lambda_k = 1/c$. Each request has the following features:

- The CU ID k, indicating it is a request made by CU_k.
- The count γ, indicating it is the γth request made by CU_k.
- Deadline D_k^γ. The deadline length of CU_k is sampled from an uniform distribution between c and D, where D is the bound of deadline lengths of all the CUs.

The overall average arrival rate to the system is then the sum of all sources $\lambda = K/c$. The offered load to the system only depends on the number of active CUs K in the scenario.

3.3 The Scheduling Policy

At the beginning of every coherence interval, the RRH sends the information of all the active requests in the ingress queue to the BBU pool. We denote the information sent by the RRH as the *report* in our model, which contains the CU ID and the deadline (k, D_k^{γ}) of all the active requests in the queue.

When the BBU pool receives a new report, it inspects the active request information in the queue and makes the corresponding *decision*, which is a set of CU IDs $\mathcal{K} \subseteq \{1, 2, 3, ...K\}$ to which the pilots are assigned. If the number of CUs with pending transmissions in the ingress queue is less than the available pilots P, it assigns all the CUs in the queue a pilot. If the number of CUs is greater than P, the EDF algorithm will be applied to allocate pilots to the P CUs whose requests have the earliest deadlines.

3.4 Fronthaul and Latency Model

The fronthaul link will cause a delay of each message sent over it. The round-trip delay of the fronthaul link is modeled as the duration from when a report departs to when the corresponding decision arrives at the RRH, but neglecting the computation time in the BBU pool for making the decision. The round-trip delay is modeled with a log-Laplace distribution with mean μ milliseconds. Our motivation for this choice is described in Section 5.1.

3.5 Performance Metrics

In this section, we detail the performance metrics: *loss* and *pilot utilization*. The pilot utilization is calculated as follows. Given a time slot j, the RRH takes a decision that $\hat{P}_j$ pilots should be assigned to the CUs in set $\mathcal{K}_j$ waiting in line, where the length of set $\mathcal{K}_j$ equals to $\hat{P}_j$, $\hat{P}_j \leq P$ and $\mathcal{K}_j \subseteq \{1, 2, 3, ...K\}$. For each CU in set $\mathcal{K}_j$, the number of transmission requests that can be served is T_{slot}/T_c, as it is proportional to the coherence interval length. We denote the actual number of active requests from $\text{CU}_k \in \mathcal{K}_j$ in the queue by $N_{k,j}$. This means that in a time slot j, the number of wasted pilots $W_{k,j}$ for CU_k is:

$$W_{k,j} = \begin{cases} 0 & \text{if } N_{k,j} \geq T_{slot}/T_c \\ \frac{T_{slot}/T_c - N_{k,j}}{T_{slot}/T_c} & \text{if } N_{k,j} < T_{slot}/T_c \end{cases} \quad (1)$$

This yields the pilot utilization in slot j:

$$U_j = 1 - \frac{\sum^{\forall k \in \mathcal{K}_j} W_{k,j}}{\hat{P}_j T_{slot}/T_c} \tag{2}$$

Taking the length of one simulation as T, the pilot utilization during the whole service period is:

$$U = 1 - \frac{\sum_{j=1}^{T/T_{slot}} \sum^{\forall k \in \mathcal{K}_j} W_{k,j}}{\sum_{j=1}^{T/T_{slot}} \hat{P}_j T_{slot}/T_c} \tag{3}$$

Denoting the actual number of requests from CU_k being served in time slot j as $S_{k,j}$, the average loss of the system during T is given by:

$$\begin{gathered} \bar{L} = 1 - \frac{\sum_{j=1}^{T/T_{slot}} \sum^{\forall k \in \mathcal{K}_j} S_{k,j}}{\sum_{k=1}^{K} \lambda_k T} \\ \text{where } S_{k,j} = \min(T_{slot}/T_c, N_{k,j}) \end{gathered} \tag{4}$$

We denote this as $\bar{L}$ because it is calculated from the mean arrival rate λ_k of each CU. In the simulation experiments, we measured the actual number of transmission requests in the system to calculate the loss L.

4 System Evaluation

In this section, we present the experiment setup and the parameter values we used in the simulations to investigate the feasibility of deploying a C-RAN system in an industrial automation scenario. The simulation is implemented in SimPy[15]. We ran all the experiments to simulate a system time of $T = 200\,000$ms and there are 20 repetitions for each parameter set.

Latency

To give an intuitive illustration of the delay incurred by the C-RAN system, we measured the round-trip delay by pinging time-stamped UDP packets from an Ubuntu 18.10 LTS machine (representing the RRH) to a remote service function hosted by a docker container residing in a virtual machine in a data-center, which is 2km away from the the RRH, representing the BBU pool.

In our simulation, we used a log-Laplace distribution to generate the round-trip delays, which, as will be shown in Section 5.1, is empirically modeled from our measurements. The mean μ of the round-trip delay varies from 0.5ms to 15ms, but the other distribution parameters remain the same for all experiments.

Table 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 2: Parameters Related to Different Mobility Scenarios in the Simulation

Mobility scenario	**High**	**Medium**	**Low**
Coherence interval length T_{slot}	0.5ms	1ms	1.5ms
Available pilots per interval P	12	24	36

Loss

To evaluate if the C-RAN system can meet the minimal requirements from the industrial standards, here, we set the maximum permissible loss to 5% for all the transmission requests.

The loss is highly related to the CUs' tolerance on the waiting time to get a radio resource, and therefore we ran experiments with the objective of investigating the maximum round-trip delay that the CUs can tolerate when they have different deadlines. We set the variables of the CUs arrival process as shown in Table 1. We choose a medium mobility scenario in this evaluation and the corresponding variables under this mobility scenario can be found in Table 2.

To invetigate the maximum number of CUs that the system can serve under different mobility scenarios, we also ran the experiments when all CUs have deadline lengths the same as their arrival intervals indicated in Table 1. We set the round-trip delay in this evaluation as 3ms, which, as will be shown in Section 5.1, is slightly larger than our latency measurements from the aforementioned experiment setup.

Pilot Utilization

The pilot utilization becomes important once the requirement of loss is met. It is obvious that the loss decreases if the scheduler allocates redundant resources to the CUs. However, this could mean that the background traffic, which has lower priority than the CU traffic, may be faced with resource starvation due to pilot waste. Thus we should consider pilot utilization under a low loss case, in which the length of deadlines has very little impact on the utilization, but the

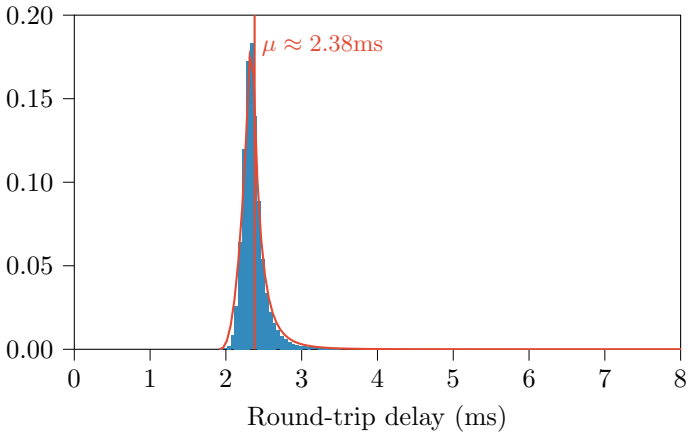


Figure 3: The histogram of the UDP round-trip delay measurements. The red curve is the probability density function and the mean value fitted from the histogram.

length of the coherence interval, or the CUs' mobility, becomes the dominating factor. Thus we ran the experiments under different mobility scenarios but with the parameters of the CUs' arrival processes the same as in Table 1, except for the deadline length, which in this case has an upper bound fixed to 15ms. The longest round-trip delay is set to 8ms, in which case there are rare discarded requests in the system for this deadline. loss

5 Experiment Results

In this section, we show our latency measurements and the simulation results regarding the two performances metrics *loss* and *pilot utilization* by following the evaluation setup.

5.1 Round-trip Delay

Figure 3 shows the histogram of our round-trip delay measurements. We fitted the histogram to a log-Laplace distribution with mean value $\mu \approx 2.38$ms. This is a long-tailed distribution, which is not just incurred by the separation between the RRH and the BBU pool, but also by the cloud execution environment.

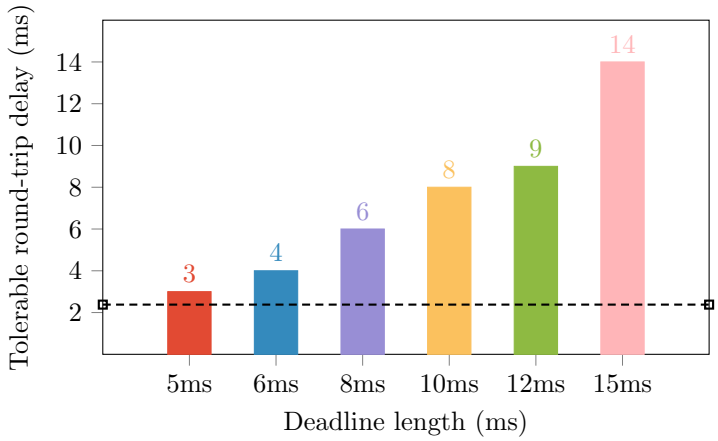


Figure 4: The tolerable round-trip delay with varying CU deadline lengths. There are in total 20 CUs, all with medium mobility. The dashed line indicates the mean round-trip delay from our measurements shown in Section 5.1.

5.2 Loss

Figure 4 shows the maximum round-trip delay the system can tolerate so that the loss is under 5% when the CUs have the arrival processes indicated in Table 1. As we can see from the Figure 4, the tolerable delay is always 1-3ms less than the deadline length. If one expects each CU to have a deadline the same length as its period, the round-trip delay incurred by the C-RAN system can not be longer than the CU's transmission interval.

Figure 5 shows the maximum number of CUs that the system can serve within the allowable loss of 5%, when all CUs have deadline length of 10ms and the round-trip delay in the system is 3ms. As can be expected, the system can serve more units when the mobility is lower (that is, when the coherence interval is longer). We can conclude from the figure that when the units have low mobility, the system can handle a higher offered load from the CUs without loss than when in a higher mobility scenario.

5.3 Pilot Utilization

Figure 6 shows how the pilot utilization is affected by the CUs mobility and the delay. When there is no delay in the system, a short coherence interval can achieve full pilot utilization. But as the interval gets longer, the utilization of the resources drops significantly to only 40% when there is only 0.5ms round-

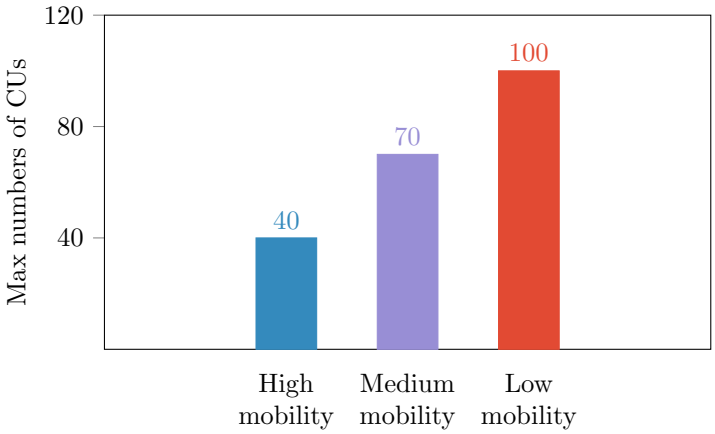


Figure 5: Maximum number of CUs the system can serve within the allowed loss of 5% under different mobility scenarios. Each CU has a deadline length of 10ms and the system round-trip delay is 3ms.

trip delay in the system. This is because when the allocation slot is shorter, the decisions are more frequently made so that they can better follow the dynamics of the ingress queue. However, having longer intervals means more time-frequency space resource are reserved for the same set of CUs in each slot. Since the transmission periods of the CUs are usually longer than the length of an allocation slot, this leads to redundant allocations when the number of the pending requests in the queue is less than which can be served by the system. However as the round-trip delay between the RRH and BBU pool increases, which may cause outdated reporting about the queuing status, the pilot utilization converges to only 10%. In this case, the length of coherence intervals has less impact, since the misreporting due to the latency causes faulty allocations, in which case a CU is allocated a pilot according to the latest arrived decision, even though all its requests were already served by previous decisions.

6 Conclusions

In this paper, we addressed the latency issue incurred by the C-RAN system characteristics and demonstrated the feasibility of deploying such a system under the industrial criteria. We considered a pilot scheduling function for industrial critical units that have stringent requirements on the deadlines. The

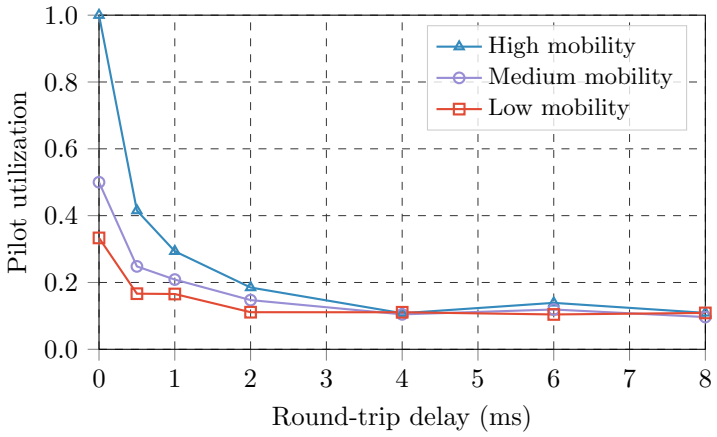


Figure 6: The pilot utilization when the number of CUs is $K = 20$ and each has a deadline length between 10 and 15ms.

function is hosted in the BBU pool but the pilots need to be allocated to CUs by the RRH. We focused on two performance metrics *loss* and *pilot utilization* and applied a simple EDF scheduling policy to evaluate if the system can cope with the delay between the scheduling function and the allocation. We performed a simulation to investigate the behavior of the system in different scenarios. Our experiment results have shown that the C-RAN system is feasible to deploy for the industrial automation scenario, where the CUs can tolerate round-trip delays up to 2ms less than their own deadlines. For a massive MIMO RRH, lower mobility end-users lead to a longer coherence interval and bring lower loss, implying that when the units' mobility is low in the scenario, the system is capable of serving a higher number of CUs simultaneously. On the other hand, both delay and a longer coherence interval lead to a huge amount of resource waste, which may lead to resource starvation of the background traffic. The next step of our work is to develop a new scheduling strategy to avoid redundant and faulty allocation so that the resources can be better utilized and the system can meet more stringent reliability requirements in industrial communication.

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation

for Strategic Research (SSF), and the 5G PERFECTA Celtic Next project funded by Sweden's Innovation Agency (VINNOVA). The authors are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

References

[1] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann. Cloud RAN for mobile networks—a technology overview. *IEEE Communications Surveys Tutorials*, 17(1):405–426, Firstquarter 2015. ISSN 2373-745X. doi: 10.1109/COMST.2014.2355255.

[2] Navid Nikaein. Processing radio access network functions in the cloud: Critical issues and modeling. In *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, MCS '15, page 36–43, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335454. doi: 10.1145/2802130.2802136.

[3] S. Mikroulis, L. N. Binh, I. N. Cano, and D. Hillerkuss. CPRI for 5G cloud RAN? – efficient implementations enabling massive MIMO deployment – challenges and perspectives. In *2018 European Conference on Optical Communication (ECOC)*, pages 1–3, Sep. 2018. doi: 10.1109/ECOC.2018.8535213.

[4] William Tärneberg. *The confluence of Cloud computing, 5G, and IoT in the Fog*. PhD thesis, Department of Electrical and Information Technology, Lund University, March 2019.

[5] Sangkyu Park, Hyunjoong Lee, Chan-Byoung Chae, and Saewoong Bahk. Massive MIMO operation in partially centralized cloud radio access networks. *Computer Networks*, 115:54 – 64, 2017. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2017.01.013.

[6] D. M. Kim, J. Park, E. De Carvalho, and C. N. Manchon. Massive MIMO functionality splits based on hybrid analog-digital precoding in a C-RAN architecture. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 1527–1531, Oct 2017. doi: 10.1109/ACSSC.2017.8335612.

[7] W. Chang, T. Xie, F. Zhou, J. Tian, and X. Zhang. A prefiltering C-RAN architecture with compressed link data rate in massive MIMO. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–6, May 2016. doi: 10.1109/VTCSpring.2016.7504095.

[8] S. Parsaeefard, R. Dawadi, M. Derakhshani, T. Le-Ngoc, and M. Baghani. Dynamic resource allocation for virtualized wireless networks in massive-MIMO-aided and fronthaul-limited C-RAN. *IEEE Transactions on Vehicular Technology*, 66(10):9512–9520, Oct 2017. ISSN 1939-9359. doi: 10.1109/TVT.2017.2712669.

[9] Y. Cai, F. R. Yu, and S. Bu. Cloud radio access networks (C-RAN) in mobile cloud computing systems. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 369–374, April 2014. doi: 10.1109/INFCOMW.2014.6849260.

[10] 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)*, pages 1–8. IEEE, feb 2018. ISBN 978-1-5386-3458-5. doi: 10.1109/ICIN.2018.8401624.

[11] China Mobile. C-RAN: the road towards green RAN, 2011. URL `https://www.semanticscholar.org/paper/C-ran-the-Road-towards-Green-Ran/eaa3ca62c9d5653e4f2318aed9ddb8992a505d3c`. Accessed on Mar. 9, 2020.

[12] Nathan J. Gomes, Philippe Chanclou, Peter Turnbull, Anthony Magee, and Volker Jungnickel. Fronthaul evolution: From CPRI to Ethernet. *Optical Fiber Technology*, 26:50–58, Dec 2015. ISSN 10685200. doi: 10.1016/j.yofte.2015.07.009.

[13] 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.

[14] ATIS White Papers. IOT categorization : Exploring the need for standardizing additional network slices. Technical Report ATIS-I-0000075, September 2019. URL `https://access.atis.org/apps/group_public/document.php?document_id=51129`. Accessed on April 19, 2020.

[15] SimPy 4.0.2. URL `https://simpy.readthedocs.io/en/latest/`. Accessed on July 13, 2020.

Paper III

Latency-aware Radio Resource Scheduling over Cloud RAN for Industry 4.0

The notion of Cloud RAN is is taking a prominent role in narrative for the next generation wireless infrastructure. Essentially, Cloud RANs offer centralized and coordinated base-band signal processing in a cloud-based BBU pool. Cloud RAN is also seen as a mean to integrated industrial communication systems. In order to provide reliable wireless connectivity for industrial deployments, by conventional means, the cloud infrastructure needs to be reliable and incur little latency, which however, is contradictory to the stochastic nature of cloud infrastructures. In this paper, we investigate the impact of stochastic delay on a radio resource scheduling process deployed in Cloud RAN. We proceed to propose a strategy for realizing timely cloud responses and then adapt that strategy to the radio resource scheduling problem. Further, we then evaluate the strategies in an industrial IoT scenario using a simulated environment. Experimentation shows a significant performance on timely responses can be achieved even with noisy cloud and that significant improvements in radio resource utilization can be attained when the strategy is applied to radio resource scheduling over Cloud RAN.

Haorui Peng, William Tärneberg, Maria Kihl, "Latency-aware Radio Resource Scheduling over Cloud RAN for Industry 4.0", submitted.

1 Introduction

The 5G is shaping the narrative for the Industry 4.0 era. With high reliability, high throughput and low latency, 5G is enabling many new applications in that domain. Further, Cloud RAN promotes softwarization and resource centralization in radio access networks and is an intriguing candidate Radio Access Network (RAN) architecture for 5G and beyond.

The basic concept of Cloud RAN is to detach the Base-Band processing Units (BBUs) from multiple legacy radio base stations and centralize them into a BBU pool, built on cloud-native techniques. The remaining Remote Radio Heads (RRHs) are only equipped with basic radio-frequency functionalities while the BBU pool allows for cooperative base-band signal processing for multiple RRH sites. In addition, more elaborate decisions and system-wide optimizations can be made when more of the system is orchestrated from the same point, such as it the case in Cloud RAN.

In a typical cloud service, a set of dynamic worker nodes are be deployed to support its workload. Then a load-balancer, distributes incoming requests to those workers. The worker nodes share virtualised resources and are subject a resource management strategy. Consequently, clouds and in extension Cloud RANs, are stochastic and dynamic systems in their own right. This so called *cloud delay* incurred by Clouds includes not only the network delays, but also the admission time and execution time. From a Cloud RAN perspective, the stochastic nature of clouds incurs detrimental delays in between signal processing functions that essentially introduce interruptions to the signal processing function chain [1].

In many future wireless systems aimed at 5G and beyond, for example, Massive Multiple Input Multiple Output (MIMO) [2], a radio resource scheduler is deployed in the RRH. The scheduler allocates the available radio resources to the User Equipments (UEs) connected to the RRH according to a policy. Often, the objective of the scheduling policy is to mitigate resource starvation, collision and congestion. When deploying such a scheduling process over Cloud RAN, the scheduling decisions are performed in the BBU pool and then actuated by the RRH.

However, the stochastic properties of the Cloud RAN environment will cause uncertainties in such a scheduling process. As the message exchanged between the BBU pool and RRH might be delayed and arrive out-of-order. In radio resource scheduling problem, this may cause false allocation and redundant allocation to the UEs. We presented in [1] the trade-offs between resource utilization and transmission reliability over the communication system when deploying a general massive MIMO radio resource scheduler over Cloud RAN. Therefore, there is a need of purpose-built schedulers that can cope with the

disturbances caused by the Cloud RAN environment.

One major challenge when deploying wireless connectivity in industrial environments is that the radio resources are shared by many units, which need ultra-reliable and low latency data transmissions for real-time control processes. It is a scenario that will benefit the most from radio resource scheduling, therefore, in this paper, we focus on Industry 4.0 application. Additionally, the stringent Quality of Service (QoS) requirements of industrial applications can act as a good benchmark also for other scenarios.

Some work has addressed the resource allocation problem for low-latency communication services in Cloud RAN under different scenarios. In [3], the authors focused on an energy consumption minimization problem for computation tasks for a mobile edge cloud enabled Cloud RAN system. Also, in [4], a energy efficient joint resource scheduling scheme is proposed for a Cloud RAN system. There are also some works like [5][6], which utilized distributed allocation algorithms to minimize the response time or the computation latency in Cloud RAN systems.

Apart from the studies on resource allocation, the characteristics of the fronthaul link latency and the jitter of the delay in Cloud RAN systems have been investigated in in [7][8].

Some papers have proposed solutions that compensate the communication delays or reduce the impact of delays for different networked systems, for example [9] and [10].

To the best of our knowledge, very few papers have addressed radio resource scheduling in a Cloud RAN environment. Further, none of these papers have dealt with delayed or out-of-order decisions.

In our work, we embrace the fact that latency over Cloud RAN systems is unavoidable and has stochastic characteristics. In this paper, we propose a Massive MIMO pilot scheduling algorithm for Cloud RANs. The proposed solution is then evaluated for an Industry 4.0 scenario with simulations. Our contributions in this paper can be summarized as follows:

- We propose a purpose-built radio resource scheduling strategy for Cloud RAN that will mitigate the impact of the cloud delay.
- We develop a simulation model for the system and evaluates the proposed solution in an Industry 4.0 scenario.
- We show that our strategy significantly improves the radio resource utilization of the system without compromising the communication reliability.

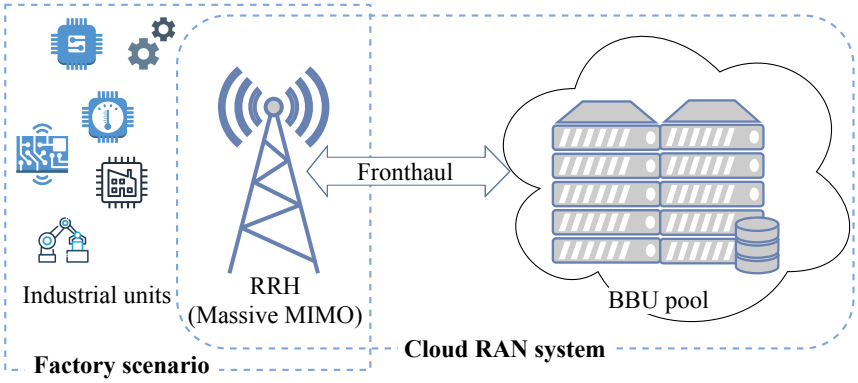


Figure 1: Target system architecture.

2 Targeted System

In this paper, we target a Cloud RAN architecture that provides wireless communications for an industrial Internet of Things (IoT) scenario. A schematic overview is given in Figure 1.

2.1 Industry 4.0 scenario

In this paper we address an indoor factory automation scenario, where industrial UEs are connected over a Massive MIMO radio network. In the envisioned industrial IoT scenario [11], the number of UEs can be extensive, with a density of 10000 devices per km^2.

We define two main types of UEs, Critical Units (CUs) and non-Critical Units (non-CUs). First, CUs are sensors, controllers, and actuators. The CUs generate control signals, usually periodically, and typically have strict QoS requirements. For example, latency less than 10ms and availability within the range of 95%-99.999%. For simplicity, we call all the signals that are exchanged within the control operations as the data transmitted via the network. Further, each transmission request from a CU has a hard deadline. If a CU was not been assigned radio capacity within the deadline, the transmission attempt failed and the data is discarded.

Second, non-CUs represent collectively other types of devices. Characteristically, they have less stringent requirements and usually sporadic transmissions. The traffic generated by non-CUs is considered as background traffic in the system.

2.2 Cloud-RAN System

A Cloud-RAN system consists of a set of RRHs connected with a BBU pool over a front-haul link. The BBU pool is deployed as a cloud-native execution environment. Consequently, the functions offered by the BBU pool are subject to stochastic delays. Also, due to opaque cloud management policies, resource scheduling, and admission, any messages sent between the RRHs and the BBU pool may come out-of-order.

Since we mainly focus on a radio resource scheduling process, which is a MAC layer function of RBS, we assume that the Physical Layer (PHY) functionalities are operated on the RRH and no raw base-band data blocks are transmitted over the front-haul link to the Cloud RAN.

For a manufacturing process, the communication distance is generally within 200m [12], thus we assume that all the UEs can be covered by the radio range of one RRH in our target scenario. However, this is not a limiting factor on our work.

2.3 Radio resource scheduling

In this paper, we adopt massive MIMO as our RAN technology. As massive MIMO is the key technology of the 5G RAN, which makes use of large-scale antenna system and promises to handle order of magnitude increasing data from numerous UEs.

We are specifically interested in *massive MIMO up-link pilot scheduling* which is one example of a resource scheduling process that can be deployed across a Cloud RAN [13, 14]. From this point on, massive MIMO up-link pilot scheduling is simply referred to as *pilot scheduling*.

The scheduler is located in the BBU pool and determines when and how to assign the pilots, based on an explicit objective. Further, the RRH allocates the up-link pilots to enable communications for the UEs of the network.

An up-link pilot is the pre-requisite for a UE to be permitted to send data during a so called coherence interval. The coherence interval is determined by for how long the wireless channel state is considered to be coherent. For example, with mobile devices, the channel state is more fleeting than with a stationary device, resulting in a shorter coherence interval. Pilot scheduling is performed for each coherence interval. If a UE is assigned a pilot, we call its transmissions can be served within the next coherence interval, and it can use the rest time-frequency space in this coherence interval for its data transmission. Irrespective of how many transmission a UE have to make, it needs at least one pilot to be allowed to transmit during an interval. If the number of UEs that have pending transmission, in a specific coherence interval, is less

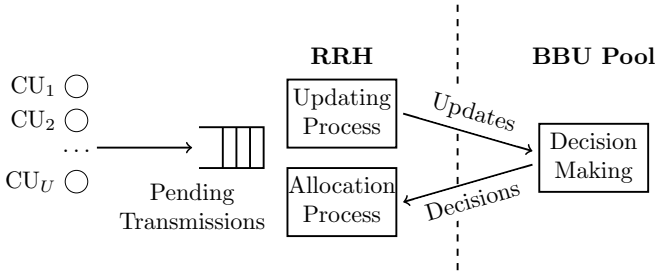


Figure 2: System model

than the number of pilots, all UEs will be permitted to transmit.

Thus the scheduling process over Cloud RAN can be divided into the following processes:

- The *allocation process* on RRH that assigns the pilots to the UEs with active pending transmission.
- 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.

3 System Model

In this section, we detail a model of the targeted system as presented in Section 2. The basic components of the system are; a set of UEs, a Cloud RAN infrastructure inclusive of a RRH and a BBU pool. Update messages are send from the RRH to the BBU in the Cloud RAN, to which the BBU responds with a scheduling decision. An overview of the system and the relation ship between those components is shown in Figure 2.

In this paper, we consider the pilot scheduling problems for the CUs, since these are the UEs with prioritized traffic. Other UEs, that is the non-CUs, will get the remaining pilots after all CUs have been served in a coherence interval.

3.1 Cloud Delay

Radio resource scheduling over Cloud RAN includes information dissemination between the RRH and the BBU, as described in Section 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 originate from the BBU pool to the allocation process at RRH.

The cloud, its opaque management systems, shared infrastructure, and intermediate network incurs a stochastic delay. This delay is represented as two independent stochastic variables, d_{update} and d_{decision}. d_{update} is the time from when an update message is sent by RRH until the message arrives at the computing unit in the BBU pool. d_{decision} is the time from when the scheduler starts to perform a decision until the decision arrives at the RRH. The two delays are inclusive of all execution times as well as admission and queuing delays in the cloud and along the path of a message.

In the following, we refer both delays to cloud delays incurred by the system.

3.2 Industrial Applications

We denote the number of active CUs covered by the radio range of the RRH, U. Further, CU_u is the uth active CU, where $u \in \{1, 2, ..., U\}$. Each CU_u triggers transmissions according to a stochastic processes to the RRH. The inter-arrival time between subsequent transmissions from CU_u is denoted c_u. A CU can only have successful transmission if it is assigned a pilot. Each transmission triggered by CU_u has a deadline D_u. A transmission is discarded and fails if it is not served by a pilot before its deadline.

3.3 Massive MIMO Scheduling

For each coherence interval, the Massive MIMO up-link pilot scheduling process allocates pilots to the resident CUs. For general applicability, a coherence interval is now referred to as a *slot*. We denote the length of a slot as T_c. For simplicity, we define that the scheduling process starts at time $t = 0$. Therefore, a slot k is the time interval $t \in [kT_c, (k+1)T_c)$ and $k = 0, 1, 2 \ldots$. Also, we assume that the BBU pool and the RRH are synchronized in time, which means that a slot k represents the same time interval at both the BBU pool and the RRH.

At the beginning of a slot k, the RRH updates the BBU about its current state, that is the number of pending transmissions from each CU_u, denoted $Q_u(k)$. In the following, we will call $Q_u(k)$ for the state of CU_u. The state of all CUs at slot k is then denoted as $\mathbf{Q}(k) = \{Q_1(k), Q_2(k)..., Q_U(k)\}$.

The BBU pool performs the scheduling decision process and then responds the RRH with the decision message, which is acted on by the RRH. We denote a scheduling decision to be applied at slot k by $\mathbf{P}(k) = \{P_1(k), P_2(k), ...P_u(k)\}$,

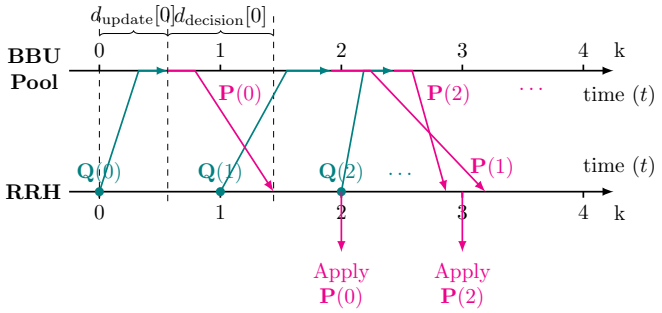


Figure 3: Sequence diagram of a naive scheduling scheme. At slot $k = 0$, an update is sent by the RRH to the BBU pool, which is successfully delivered after $d_{\text{update}}[0]$. A scheduling decision $\mathbf{P}(0)$ is made upon the arrival of this update and is sent to the RRH, however it may arrive later than it is meant to be applied due to the delay.

where

$$P_u(k) = \begin{cases} 1 & \text{pilot assigned to CU}_u\text{at slot } k \\ 0 & \text{no pilot assigned to CU}_u\text{at slot } k \end{cases} \tag{1}$$

At every slot k, the RRH allocates pilots to the active CUs according to the decision $\mathbf{P}(k)$. We define that, in total, p pilots are available per slot. Consequently, at most p CUs can be assigned pilots per slot. If $P_u(k) = 1$, N transmissions from CU_u can be served at slot k. The time $t = kT_c$ is also called as the actuation time of a decision, and k is the underlying actuation slot.

4 Problem Definition

In this section, we detail the challenges incurred by the stochastic properties of a Cloud RAN system on the scheduling process. We begin with describing the main obstacles when a *naive pilot scheduling scheme* is deployed to a Cloud RAN. Here, the scheduler is triggered every time an update message is delivered to the BBU pool. Upon completion, a scheduling decision is sent to the RRH. The inherent delay over the Cloud RAN infrastructure is not accounted for in the scheduler. This dynamic is shown in Figure 3.

As we saw from Figure 3 that, without taking into account the stochastic delays of update messages and scheduling decisions, a decision $\mathbf{P}(k)$, which is a response to an update message $\mathbf{Q}(k)$, may fail to be actuated at slot k. This

would lead to false allocation and redundant allocation in a pilot scheduling process, and further results in unwanted performances as we discovered in [1].

We herein consider the following three performance metrics: *timely applied decisions*, *loss*, and *pilot utilization* to examine whether a scheduling process performs properly across Cloud RAN system. Below we define a set of challenges based on these performance metrics.

4.1 Timely Applied Decisions (R)

The cloud delay may cause a decision to arrive later than the slot it should be actuated in as we she in Figure 3. In this case, the decision is considered outdated and not timely applied. Conversely, at slot k, the decision $\mathbf{P}(k)$ is applied, we call this timely applied decision at k.

The ratio between timely applied decisions and all decisions is denoted R. Also, we denote $R_{k_i:k_j}$ as the ratio of timely applied decisions from slot k_i to k_j. Suppose $\mathcal{K}$ is the number of decisions timely applied between slots k_i and k_j:

$$R_{k_i:k_j} = \frac{\mathcal{K}}{k_j - k_i} \tag{2}$$

When a decision is applied at a slot it is not intended to be, the state of pending transmissions may deviate, further false allocation or redundant allocation may occur, leading to performance degradation in *loss* and *pilot utilization* in the case of pilot scheduling.

4.2 Loss (L) and Pilot Utilization (β)

We use *loss*, denoted L, and *pilot utilization*, denoted β, to evaluate the performance of the pilot scheduling strategy.

Low loss is crucial for time-critical industrial applications. Every time, a CU triggers a new transmission that needs to be served within its deadline. Otherwise, the transmission expires and is discarded. The loss of the radio system, L, can be directly evaluated at the RRH and is defined as the ratio of expired versus the total number of transmissions.

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 send. This can occur if a scheduling decision is based on an outdated RRH state and if scheduling decisions are not delivered timely. We assume that the decision $\mathbf{P}(k)$ determines a set of CUs, $\mathcal{U}$, that are allocated pilots, where $\mathcal{U} \subseteq \{1, 2, 3, ...U\}$ and the size of set $\mathcal{U}$ is less than or equal to p. For each CU in set $\mathcal{U}$, the number of transmissions that can be served is N. However the

actual number of pending transmissions from CU_u at this moment is $Q_u(k)$. This will yield the following number of wasted pilots, denoted $\omega_u(k)$, for CU_u at slot k:

$$\omega_u(k) = \begin{cases} 0 & \text{if } Q_u(k) \geq N \\ \frac{N - Q_u(k)}{N} & \text{if } Q_u(k) < N \end{cases} \tag{3}$$

which yields the pilot utilization $\beta(k)$ for all CUs for this allocation:

$$\beta(k) = 1 - \frac{\sum^{\forall i \in \mathcal{U}} \omega_u(k)}{\texttt{sizeof}(\mathcal{U}) * N} \tag{4}$$

4.3 Research challenges

Now that we have a model and defined the performance metrics of the scheduler we can begin the discuss the inherent challenges with radio resource allocation over Cloud RAN. Firstly, we can now define the objective of the scheduler as follows:

- Assign pilots to all CUs with pending transmissions, in a fair manner, before their transmissions have expired.
- While avoiding starving the background traffic, which is a consequence of resource waste.

We have shown in [1] and as illustrated in Figure 3, that a naive scheduling method over Cloud RAN, is feasible when meeting the reliability requirements for industrial standards, by keeping the loss under 5%. However, a naive scheduling would also lead to huge amounts of pilot waste. Therefore, in the next section, given the stochastic nature of Cloud RAN, we propose a new scheduling strategy for pilot scheduling over Cloud RAN that is focused on improving the pilot utilization for time-critical applications without compromising transmission reliability, that is keeping the loss under 5%.

5 Proposed Solution

In this section, we propose an novel massive MIMO up-link pilot scheduling strategy over Cloud RAN that addresses the challenge of timely arrival of decisions, as detailed in Section 4. Our proposed solution can handle the delayed

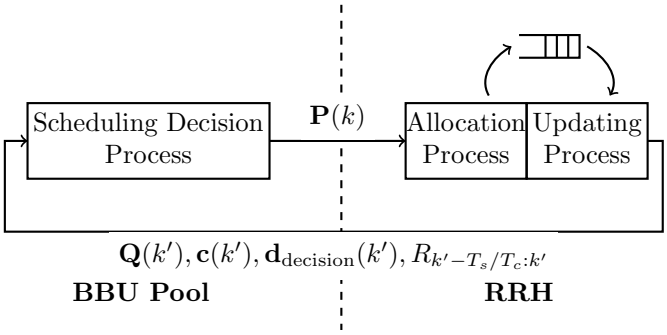


Figure 4: Overview of the scheduling process over Cloud RAN at slot k'. The RRH first sends an update message to the BBU pool (*updating process*). Secondly, it allocates the pilots according to the arrived scheduling decision $\mathbf{P}(k')$ (*allocation process*) . At the same time the BBU pool performs a decision for a specific future slot k and sends this decision to the RRH (*scheduling decision process*).

and out-of-order messages that is an effect of a stochastic Cloud RAN environment. Thereby, the pilot utilization is radically improved, without compromising reliability performance. An overview of the proposed solution is shown in Figure 4.

Our proposed solution makes use of the following concepts:

1. **Save and sort:** In *allocation process*, to remedy out-of-order arrivals, the RRH saves and sorts incoming scheduling decisions and applies them at their intended actuation time. If duplicate decisions are received for a slot, the most recent arrival is applied. Further, decisions that arrive after their intended actuation slot are discarded.

2. **Fallback:** In *allocation process* at the RRH, if a scheduling decision is absent for a slot, the nearest decision, in time, is applied.

3. **Strategic delay of decisions:** To remedy stochastic delay, the *updating process* and *scheduling decision process* are handled asynchronously. The state of the RRH in time is estimated in *scheduling decision process* by making use of the contributes from *updating process*. Upon arrival, an update does not trigger a response from the scheduling decision, instead, decisions are generated periodically based on the estimated state of the RRH, at a set frequency. The scheduling decisions are bound for a particular point in time, a horizon. The horizon, can be set so that scheduling

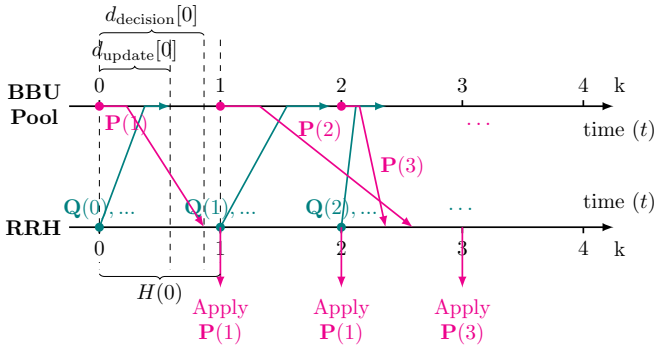


Figure 5: Sequence diagram of the scheduling process with our proposed solution. At slot $k = 2$, a decision $\mathbf{P}(3)$ is made based on the the latest arrived queue size update $\mathbf{Q}(1)$. The decision is indented to be applied at slot $k = 3$.

decisions arrive at the BBU with a set probability, given the cloud delay. Consequently, decisions from the BBU will arrive and applied timely at the RRH with a desired probability.

4. **Redundancy:** To accommodate for lost messages and be able to control the horizon, without sacrificing performance, redundancy is introduced. In the *scheduling decision process*, the BBU saves and includes a set of past and future scheduling decisions in each outbound decision.

Fig. Figure 5 shows a sequence diagram of our proposed strategy, to be contrasted with Figure 3. Below we describe the details of our proposed scheduling strategy.

5.1 Updating Process

At each slot k', the RRH sends an update message to the BBU, for the purpose of state estimation of the RRH in the BBU pool. The update message includes the following:

- The number of pending requests $\mathbf{Q}(k')$ for each user.
- The inter-arrival times of the transmissions from each CU $\mathbf{c}(k') = \{\mathbf{c}_1(k'), \mathbf{c}_2(k'), ...\mathbf{c}_U(k')\}$, which arrived during slot $k' - 1$. Here, $\mathbf{c_u}(k')$ is the set of all inter-arrival samples of CU_u measured during slot $k' - 1$, thus $\mathbf{c}_u(k') = \{c_u[n_1], c_u[n_2], ...\}$.

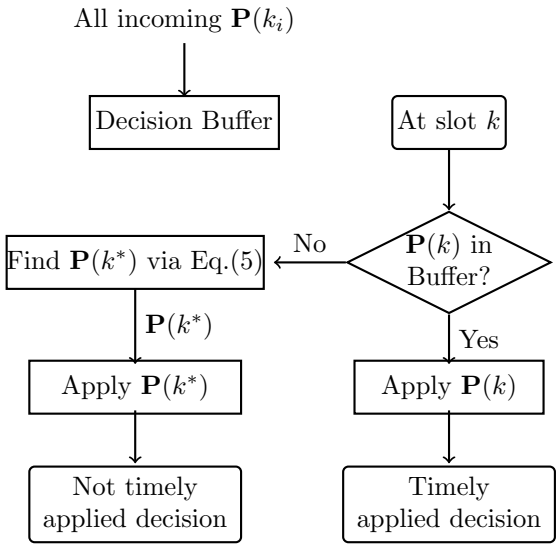


Figure 6: Allocation process at RRH

- The measured delay samples from scheduling decision messages that have arrived during slot $k' - 1$. $\mathbf{d}_{\text{decision}}(k') = \{d_{\text{decision}}[m_1], d_{\text{decision}}[m_2], ...\}$.

Further, every T_s, the RRH includes timely applied decisions during last T_s in the update message. The timely applied decision is noted as $R_{k'-T_s/T_c:k'}$ if sent at slot k'. This information contributes to the horizon prediction in the scheduling decision process.

5.2 Allocation Process

At each slot k, after sending the update message, the RRH applies a received decision to assign the pilots to the active CUs. The allocation process is detailed in Figure 6.

As the decisions performed by the BBU are intended to be actuated in specific slots, there needs to be a solution for decisions that arrive delayed or out-of-order. Therefore, we propose that the RRH buffers all arrived decisions and applies them at the intended actuation slot.

If one decision fails to be delivered before its intended applied slot, the RRH takes a buffered scheduling decision $\mathbf{P}(k^*)$ that is intended for slot k^* via Equation (5) and applies this decision instead. This is based on the assumption

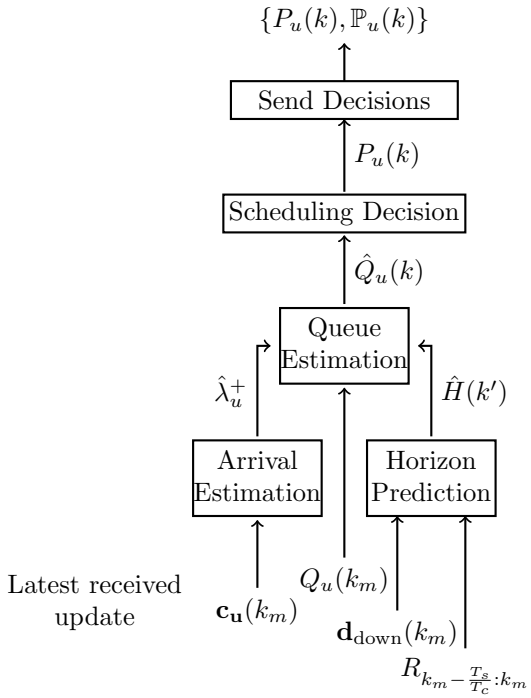


Figure 7: Scheduling Decision process in the BBU pool at slot k', taking into account the updates sent by the RRH at time k_m, which is the nearest slot to k' among all the arrived updates. The process performs a decision expected to be applied by the RRH at slot k, where $k \geq k'$

that the state estimation for the nearest slot will, on average, is the second most accurate.

$$k^* = \underset{k_i}{\text{argmin}} \, |k_i - k|, \; \forall \mathbf{P}(k_i) \text{ in decision buffer} \tag{5}$$

5.3 Scheduling Decision Process

The scheduling decision process in our proposed solution can be divided into several sequential sub-processes as presented Figure 7. In the following, we detail every sub-process as illustrated in the figure.

Scheduling Decision

The scheduler performs a scheduling decision, $\mathbf{P}(k)$, to be applied at a future slot k. The decision is based on the estimated state of the RRH on all active CUs at slot k.

In this paper, we implement a greedy allocation strategy, however, other scheduling methods can of course be used. The decision for CU_u is, $P_u(k) = 1$ if $Q_u(k)$ is non-zero and has one of the p largest values among the set $\mathbf{Q}(k)$.

The scheduling decision message includes both the newly made decision $\mathbf{P}(k)$ and h redundant scheduling decisions $\mathbb{P}(k)$, where the intended actuation slots are before k:

$$\mathbb{P}(k) = \{\mathbf{P}(k-1), \mathbf{P}(k-2), ...\mathbf{P}(k-h)\} \tag{6}$$

Using redundant messages means that if a decision intended for slot k is delayed and thereby arrives later than its intended actuation slot, later decision messages may be able to deliver this decision for slot k in time to the RRH. This not only significantly improves the timely applied decisions, but also benefits the state estimation accuracy, as it will be shown in the results.

Queue Estimation

The decision $\mathbf{P}(k)$ is indented to be applied at a future slot k. Thereby a state estimation at k needs to be provided, which is denoted by $\hat{\mathbf{Q}}(k)$.

Considering that at slot k', we take the state $Q_u(k_m)$ of CU_u from all the received update messages at the BBU pool, where $k_m \leq k'$ and:

$$\begin{aligned} &k_m = \max\{k_i\} \\ &\forall \mathbf{Q}(k_i) \text{ in the received update messages} \end{aligned} \tag{7}$$

If the average arrival rate of the requests from CU_u is λ_u, and the predicted time horizon for when a decision should be actuated is $H(k')$, the queue sizes for slot k, $\hat{Q}_u(k)$, can be estimated queue as follows:

$$\begin{aligned} &\hat{Q}_u(k) = Q_u(k_m) + \lambda_u(k - k_m) - \sum_{\kappa=k_m}^{k-1} P_u(\kappa) \\ &\text{where } k = k' + H(k') \end{aligned} \tag{8}$$

The term $\sum_{\kappa=k_m}^{k-1} P_u(\kappa)$ corresponds to all decisions that are presumably to be applied from slot k_m to $k-1$.

Arrival Process Estimation

In Equation (8), the term $\lambda_u(k - k_m)T_c$ is used to predict the number of transmissions for CU_u that have been triggered during $[k_m T_c, kT_c)$. We use an Exponential Moving Average (EMA) estimator in order to estimate average inter-arrival time $\hat{c}_u$ of requests for CU_u, which gives:

$$\hat{c}_u^+ = \alpha_c \hat{c}_u^- + (1 - \alpha_c)\bar{c}_u \tag{9}$$

Here, $\bar{c}_u$ is taken from the inter-arrival time sample $\mathbf{c_u}(k_m)$ informed in the most update message, wherein k_m is located by Equation (7). We denote by $\hat{c}_u^+$ the new estimate on c_u. $\hat{c}_k^-$ is the old estimate and α_c is the weight of the EMA estimator. Further, the average arrival rate of CU_u can trivially be derived as:

$$\hat{\lambda}_u^+ = 1/\hat{c}_u^+ \tag{10}$$

Predicted Time Horizon

A decision message is performed in slot k' and should be applied in slot k, where $k \geq k'$. $k - k'$ is defined as the predicted time horizon, $\hat{H}(k')$. The predict time horizon is a crucial part of our proposed strategy, since it determines how delayed a decision message can be. A longer predicted time horizon will increase the ratio of timely applied decision, however, at the same time introduce more inaccuracies in the stare estimation.

Therefore, in this paper, we propose to calculate the predicted time horizon by using an estimate of the average time from when the scheduler starts to perform a decision until the decision arrives at the RRH, $\hat{d}_{\text{decision}}$, and adding an offset σ, as follows:

$$\hat{H}(k') = \left\lceil \frac{\hat{d}_{\text{decision}}^+}{T_c} \right\rceil + \sigma^+ \tag{11}$$

Here, $\hat{d}_{\text{decision}}^+$ is the estimation of the average decision delay given by an EMA with weight α_d:

$$\hat{d}_{\text{decision}}^+ = \alpha_d \hat{d}_{\text{decision}}^- + (1 - \alpha_d)\bar{d}_{\text{decision}} \tag{12}$$

Similar to the average inter-arrival time estimator in section Section 5.3, $\bar{d}_{\text{decision}}$ is a sample of the decision delay, informed in the update message $\mathbf{d}_{\text{decision}}(k_m)$. $\hat{d}_{\text{decision}}^-$ is the previous estimate of the average decision delay.

The offset value σ^+ is an output of a step controller via Equation (13) when a new update on the average timely applied decision ratio $R_{k_m - \frac{T_s}{T_c}:k_m}$ has arrived.

$$\sigma^{+} = \begin{cases} \sigma^{-} + 1 & \text{if } R_{k_m - \frac{T_s}{T_c}:k_m} < r \\ \sigma^{-} & \text{Otherwise} \end{cases} \tag{13}$$

Here, σ^{-} is the previous offset value. In this paper, σ^{-} is initialized as 0, however, this is not a limiting factor.

The feedback $R_{k_m - \frac{T_s}{T_c}:k_m}$ should be calculated from a sequence of past slots and the measurements size should be large enough to be confident. We thus define the sampling time of the step controller as T_s, which is much greater than the scheduling time slot length $T_s \gg T_c$. Therefore, $R_{k_m - \frac{T_s}{T_c}:k_m}$ is collected through every T_s/T_c scheduling slots. In this way, the mean estimation on $\hat{d}_{\text{decision}}$ is made every T_c but σ is made every T_s.

The lower bound reference value for timely applied decisions ratio is r and the predicted time horizon is increased by 1 slot if the reported ratio of timely applied decisions is lower than r. In this way, if the number of discarded decisions exceeds a set point, the predicted time horizon is extended by increasing the offset value. This means that the probability that a decision arrives before its indented actuation time is increased.

If a decision is performed at slot k', the indented actuation slot k is easily given as follows:

$$k = k' + \hat{H}(k') \tag{14}$$

6 Experiments

In this section, we describe our experiments for evaluating the performance of our proposed pilot scheduling strategy over Cloud RAN. In our evaluation, we address the three performance metrics described in section Section 4, i.e timely applied event, loss of transmissions, and pilot utilization. We examine how these performance metrics are affected by the stochastic properties of a Cloud RAN.

We evaluated our proposed strategy in a simulated environment built on SimPy [15] that uses the system model described earlier. We ran all experiments for a simulated system time of $T = 200$s and the results are based on the average of 20 repetitions. As a result, all confidence intervals are within 10% of the corresponding average value.

6.1 Simulation Parameters

The system model includes several system parameters that need to be set. These are described below.

Table 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 length	10ms	D_u

Table 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	$p(x) = \frac{1}{2\mu_1} e^{-\frac{x}{\mu_1}} + \frac{1}{2\mu_2} e^{-\frac{x}{\mu_2}}$ $\mu_1 = \mu(1 + \frac{\sqrt{2}}{2}),\ \mu_2 = \mu(1 - \frac{\sqrt{2}}{2})$

Arrival process of transmissions

To generate traffic that can correspond to time critical industrial applications, we use the industry and IoT traffic models summarized in [12]. Each CU_u generates transmissions according to a homogeneous periodic stochastic process, with inter-arrival time $c_u \sim \mathcal{N}(c, \delta^2)$. Table 1 lists all parameters related to the arrival process of the transmissions and the values used in our simulations.

Stochastic delay

In this paper, we use the exponential distribution family to generate the two parameters representing the cloud delay, d_{update} and d_{decision}. For all distributions, the average delay was μ. We examine how different delay distributions and μ affect the system performance.

In the simulations, we evaluated the system performance when the cloud delay distribution is deterministic, Erlang distributed, Exponential distributed, and Hyper-exponential distributed. We chose parameters so that the coefficient of variance, CV^2, was $\{0, 0.5, 1, 2\}$. The probability density functions used in our simulation are shown in Table 2.

The average delay, μ, was varied from 0ms to 4ms, where $\mu = 0$ represents a system with a scheduler co-located with the RRH.

Table 3: Parameters of the Allocation Process in the RRH

Parameter name	**Value**	**Symbol**
Scheduling time slot length	0.5 ms	T_c
Number of available pilots per slot	12	p
Number of requests served by a pilot	1	N

Table 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_s
Redundant Decisions	Number of redundant decisions	2	h

Scheduling strategy

Our proposed scheduling strategy includes several parameters, and in this section we detail the values used for these parameters in the simulations.

The values of the parameters in the allocation process, placed in the RRH, are shown in Table 3. The values of these parameters correspond to the radio spectrum parameters of our massive MIMO test-bed[16].

Table 4 lists the values for the parameters used in the scheduling decision process placed in the BBU pool.

6.2 Evaluation Methods

The objective of the evaluation is to show that our proposed pilot scheduling strategy efficiently mitigates the negative effects of the stochastic properties of the Cloud RAN, and thereby improves the pilot utilization without compromising reliability performance. Since such a strategy needs to mitigate the delayed and out-of-order decision messages, we will in the result section show how our proposed solution performs in comparison with three other methods that do not include the full set of remedy strategies. The strategies we used in the evaluation and their corresponding system parameters are summarized in Table 5. The details for the naive scheduling method under the same scenario is studied in [17].

In the experiments, we refer to the 95% availability industrial requirement noted in Section 2.1 and set the maximum permissible loss to 5% for all the

Table 5: Evaluated Scheduling Strategies in the Experiments

Method Name	**Parameters**
Proposed Solution	Indicated in TABLE Table 4
Single Decision	Same as Proposed Solution but $h = 0$
Short Horizon	Same as Proposed Solution but $h = 0, \sigma \equiv 0$
Naive Scheduling	Described in Fig.Figure 3

transmissions, then examine the pilot utilization performance when this condition is satisfied.

7 Results

In this section, we present and discuss our simulation results. We show that our proposed scheduling strategy of increasing the number of timely applied decisions, significantly improves pilot utilization, while meeting the industrial reliability requirement as noted in Section 2.1. Below we first present the results of our strategy for timely applied decisions. Then we present the results of the pilot scheduling process that relies on the ratio of timely applied decisions.

7.1 Timely Applied Decisions

With a high proportion of timely applied decisions, the scheduling strategy has been able to successfully mitigate the adverse effects of a Cloud RAN system. As a reference point, Figure 8 shows that when a naive scheduler is employed, no decision will be timely applied. This is because, the naive scheduler does not take into account the cloud delays incurred in the system, all decisions will arrive later than their intended actuation slot.

An extended predicted time horizon can improve the ratio of timely applied decisions, as revealed by the comparison between Naive Scheduling, Short Horizon and Single Decision in Figure 8, wherein the prediction horizon increases in turn. However this result is rather logical, a longer predicted time horizon means that decisions can experience a longer delay without arriving too late. This means decisions will on average arrive before their intended actuation slot, and thereby be timely applied. However, there is a trade-off between the prediction time horizon and the accuracy of the state estimation.

Further, Figure 8 shows that when adding redundant messages, the ratio of timely applied decisions is also improved, for all experiments. As the example we illustrate in Figure 9 that, by having redundancy, the Proposed Solution

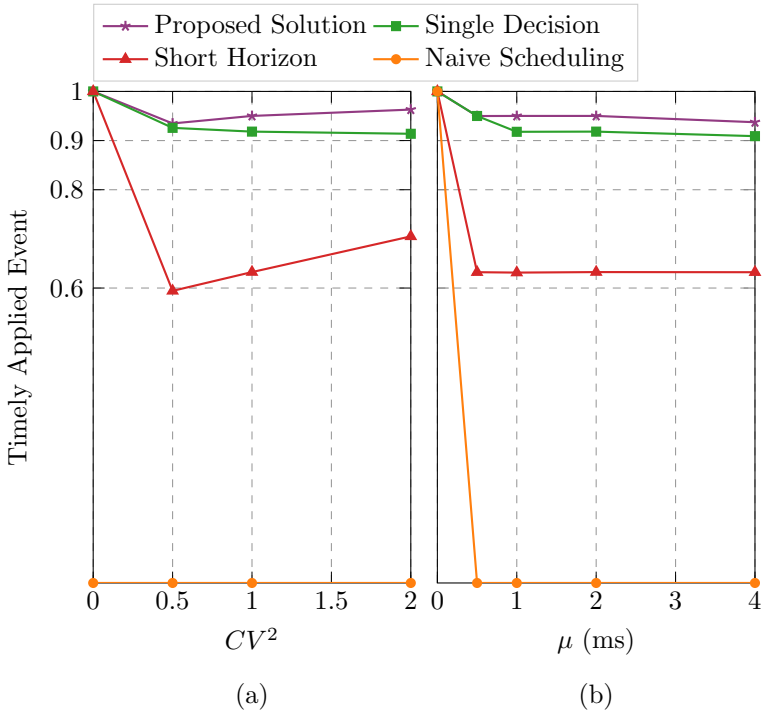


Figure 8: Timely applied decision for the four methods (a) under different delay distributions when μ=2ms and (b) when μ increases for exponentially distributed delays.

can reach over 90% timely applied decisions without a large prediction horizon. This further compensates the trade-off between the horizon and state estimation accuracy. In Figure 9, we show the response plots of timely applied decisions and underlying prediction horizon as the delay distribution changes over time. We see that, with a shorter horizon compare to the Single Decision method, having redundancy in Proposed Solution contributes a more robust and faster response to keep the timely applied decision above the reference value.

Fig. Figure 8 also shows that the ratio of timely applied decision is not greatly affected by the length of the average delay. Furthermore, a larger variance in the distribution may even improve the timely applied decisions. This result is mainly an effect of the different distributions we have used for the cloud delays. For the hyper-exponentially distributed delays with density

function in Table 2, the probability that $d_{\text{decision}} \leq \mu$ is higher than for the other distributions.

Show in Figure 10 is an example empirical Cumulative Distribution Function (CDF) plot of the hyper-exponential distributed delay used in the experiments. The figure illustrates the relationship between the length of prediction horizon and the timely applied decisions. If we use the estimated mean value of the delays as the prediction horizon (Short Horizon method), only around 71% decisions will arrive before their assigned slot. For 90% of decisions to be applied timely, an offset of more than 3ms in the case should be added to the estimated mean value as the prediction horizon, according to the CDF.

Therefore, the ratio of timely decisions is highly correlated to the delay distribution and the length of perdition horizon. In this paper. We make use of the strategy detailed in Section 5.3 to determine the prediction horizon. But it is an open question and various methods can be adopted to make the prediction.

7.2 Pilot Utilization

Fig. 11 shows the resulting average pilot utilization for our proposed up-link pilot scheduling strategy, compared with pilot utilization when using a naive greedy allocation scheduling strategy, as described earlier. We note that with both our Proposed Solution and the Naive Scheduling the loss of the transmissions is below 5%, as is required by the industrial standards. The Naive Scheduling method meets the transmission deadlines by keeping assigning redundant pilots to serve a single transmission. Although the Single Decision and Short Horizon methods significantly improved the timely applied ratio comparing to the naive scheduling method, the loss with these two methods does not meet the industrial standards, as the increment is not high enough to compensate the inaccuracy in state estimation.

Comparing to a naive scheduling method, our proposed pilot scheduling strategy increases the pilot utilization from less than 20% to over 90%. This means that our proposed strategy effectively mitigates the stochastic delays and out-of-order messages, which are the main effects of the Cloud RAN system. When less pilots are wasted on the CUs, the system becomes more capable to serve the traffic from non-CUs, which means that starvation of these applications can be avoided.

Comparing Figure 11 and Figure 8, it is clear that the pilot utilization is considerably impacted by the ratio of timely applied decisions. Basically, when more decisions are timely applied, less pilots are wasted. But we also see that the utilization is not completely decided by the timely applied decisions, but also the mean delays. As longer delay yields longer prediction horizon, which

leads to more inaccuracies in the state estimation of the RRH.

8 Conclusions

In this paper, we have investigated how scheduling can be performed over Cloud RAN. We have focused on the stochastic characteristics incurred by the Cloud RAN. We have proposed a massive MIMO up-link pilot scheduling strategy that mitigates the impacts of the Cloud RAN, in particular the stochastic delays and out-of-order messages. We have evaluated our proposed strategy with simulations. The effects of the Cloud RAN are mainly mitigated by including a predicted time horizon and sending redundant decisions, which are used to perform a scheduling decision for a future time slot.

Our experiment results have shown that the proposed strategy significantly improves the pilot utilization by increasing the ratio of timely applied decisions, without compromising the industrial requirements on transmission reliability.

Redundancy will definitely be an important part of our scheduling strategy over Cloud RAN, and this is a completely new idea compared with previously published networked schedulers. In this paper, we have not tried to optimize the number of redundant decisions, just showing the advantageous of including them. However, the optimal number of redundant decisions will of course depend on the available bandwidth of the front-haul link and the length of the cloud delay.

We also show that there is a trade-off between the length of the predicted time horizon and the accuracy of state estimation. In this paper, we have not investigated this trade-off, however, this should of course be performed in a future work. Furthermore, we would also like to address an networked test-bed and deploy the scheduler in a real cloud environment to evaluate how the proposed strategy can deal with a more practical and complicated execution environment. A more concrete delay characterization of a cloud environment would be considered to develop an optimal scheduling strategy for such a system.

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the 5G PERFECTA Celtic Next project funded by Sweden's Innovation Agency (VINNOVA). The authors are part

of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

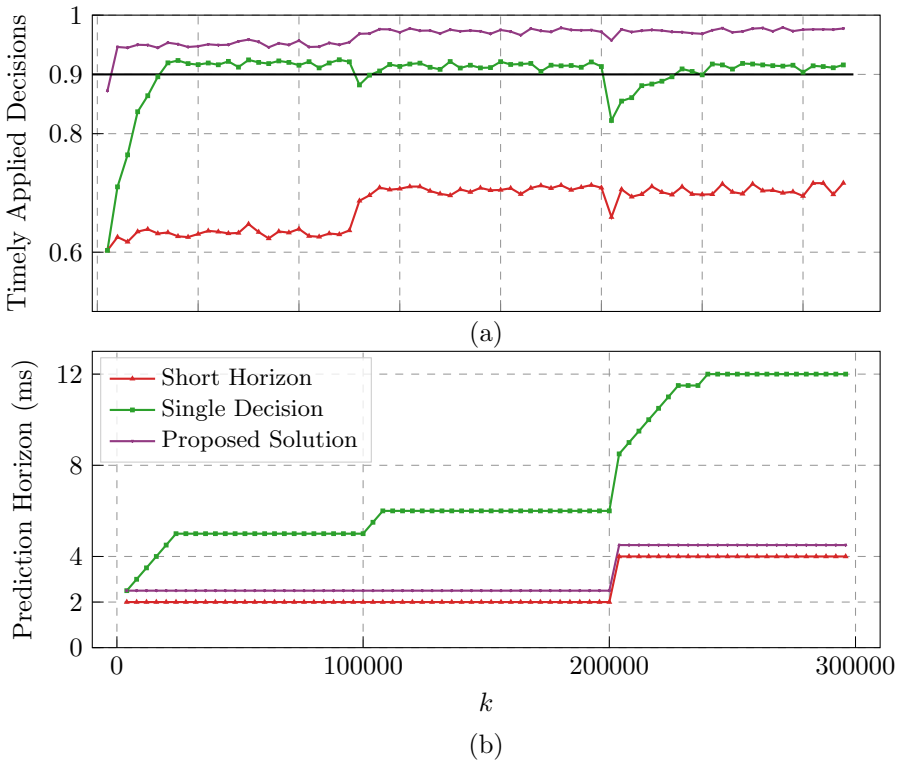


Figure 9: Response plots of (a) timely applied decisions (b) prediction horizons with different methods as decision delay distribution changes over time. During $k \in [0, 10000)$, decision delay $\mu = 2$ms, $CV^2 = 1$; During $k \in [100000, 200000)$, decision delay $\mu = 2$ms, $CV^2 = 2$; During $k \in [200000, 300000)$, decision delay $\mu = 4$ms, $CV^2 = 2$.

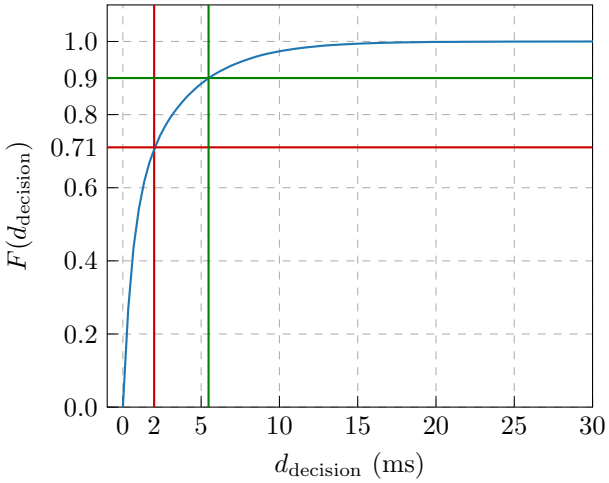


Figure 10: The empirical CDF plot of decision delay samples when $\mu = 2$ms and $CV^2 = 2$ (a hyper-exponential distribution). The red line shows the cumulative probability $P(d_{\text{decision}} \leq \mu)$. The green line shows value of decision delay at which the cumulative probability is 0.9, which is the lower bound reference value of the step controller used in delay horizon prediction.

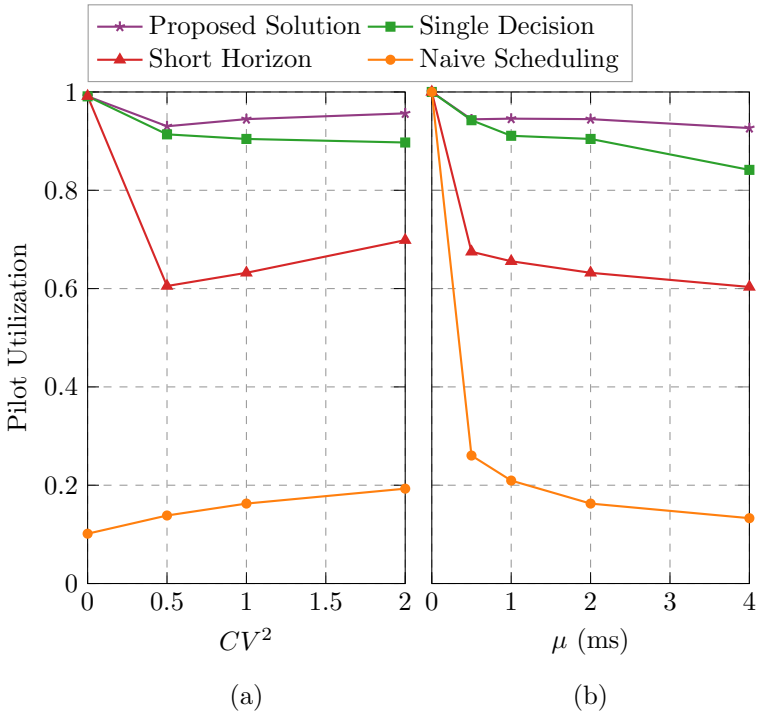


Figure 11: Pilot utilization (a) under different delay distributions when μ=2ms and (b) as μ increases for exponentially distributed delay

References

[1] Haorui Peng, William Tärneberg, Emma Fitzgerald, and Maria Kihl. Massive MIMO pilot scheduling over Cloud RAN for Industry 4.0. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, sep 2020.

[2] Erik G. Larsson, Ove Edfors, Fredrik Tufvesson, and Thomas L. Marzetta. Massive MIMO for next generation wireless systems. *IEEE Communications Magazine*, 52(2):186–195, Feb 2014. ISSN 0163-6804. doi: 10.1109/MCOM.2014.6736761.

[3] 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. *IEEE Internet of Things Journal*, 7(4): 3282–3299, apr 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.2967502.

[4] 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)*, pages 1–6. IEEE, dec 2016. ISBN 978-1-5090-1328-9. doi: 10.1109/GLOCOM.2016.7841793.

[5] 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*, volume 2018-Janua, pages 1–6. IEEE, dec 2017. ISBN 978-1-5090-5019-2. doi: 10.1109/GLOCOM.2017.8254627.

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

[7] Jobin Francis, Jay Kant Chaudhary, Andre Noll Barreto, and Gerhard Fettweis. Uplink Latency in Massive MIMO-Based C-RAN With Intra-PHY Functional Split. *IEEE Communications Letters*, 24(4):912–916, apr 2020. ISSN 1089-7798. doi: 10.1109/LCOMM.2020.2966612.

[8] Divya Chitimalla, Koteswararao Kondepu, Luca Valcarenghi, Massimo Tornatore, and Biswanath Mukherjee. 5G Fronthaul–Latency and Jitter Studies of CPRI Over Ethernet. *Journal of Optical Communications and Networking*, 9(2):172, feb 2017. ISSN 1943-0620. doi: 10.1364/JOCN.9.000172.

[9] Yan Zong, Xuewu Dai, Pep Canyelles-Pericas, Krishna Busawon, Richard Binns, and Zhiwei Gao. Modelling and Synchronisation of Delayed Packet-Coupled Oscillators in Industrial Wireless Sensor Networks. apr 2020.

[10] Da Xue and Nael H. El-Farra. Optimization-Based Actuator and Communication Scheduling in Networked Distributed Processes with Communication Delays. In *2019 American Control Conference (ACC)*, volume 2019-July, pages 2558–2563. IEEE, jul 2019. ISBN 978-1-5386-7926-5. doi: 10.23919/ACC.2019.8814477.

[11] ATIS White Papers. IOT categorization : Exploring the need for standardizing additional network slices. Technical Report ATIS-I-0000075, September 2019. URL `https://access.atis.org/apps/group_public/document.php?document_id=51129`. Accessed on April 19, 2020.

[12] 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)*, pages 1–8. IEEE, feb 2018. ISBN 978-1-5386-3458-5. doi: 10.1109/ICIN.2018.8401624.

[13] Emil Bjornson, Erik G. Larsson, and Merouane Debbah. Massive MIMO for maximal spectral efficiency: How many users and pilots should be allocated? *IEEE Transactions on Wireless Communications*, 15(2):1293–1308, feb 2016. ISSN 1536-1276. doi: 10.1109/TWC.2015.2488634.

[14] Emma Fitzgerald, Michal Pioro, and Fredrik Tufvesson. Massive MIMO Optimization With Compatible Sets. *IEEE Transactions on Wireless Communications*, 18(5):2794–2812, May 2019. ISSN 1536-1276. doi: 10.1109/TWC.2019.2908362.

[15] SimPy 4.0.2. URL `https://simpy.readthedocs.io/en/latest/`. Accessed on July 13, 2020.

[16] 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. *IEEE Access*, pages 9073 – 9088, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2705561.

[17] Haorui Peng, William Tärneberg, Emma Fitzgerald, and Maria Kihl. Massive MIMO pilot scheduling over Cloud RAN. In *16th Swedish National Computer Networking Workshop (SNCNW 2020)*, may 2020.

Paper IV

FedApp: a Research Sandbox for Application Orchestration with Kubernetes on the Next-Generation Cloud and Edge Computing Infrastructures

Multi-cluster federation is envisioned to be the next-generation cloud infrastructure, where it will play a vital part in the realization of concepts such as edge clouds. Orchestrating applications in these environments pose new challenges to well-known research problems in networking and automatic control, such as load-balancing and auto-scaling. However, as access to real multi-cluster infrastructure is limited, a testbed that provides similar characteristics to a real system is in demand. To enable researchers in these fields to quickly setup and experiment in a federated cluster environment, we have created the open-source sandbox FedApp that simplifies the process of deploying multiple virtual clusters in an OpenStack environment with the possibility of adding realistic network characteristics between sites. Each cluster comes deployed with the open-source and production-grade container orchestrator Kubernetes, complete with federation-wide monitoring using Prometheus/Grafana and simplified inter-cluster microservice communication using Istio.

Johan Ruuskanen, Haorui Peng, Alfred Åkesson, Lars Larsson, Maria Kihl, "FedApp: a Research Sandbox for Application Orchestration with Kubernetes on the Next-Generation Cloud and Edge Computing Infrastructures", to be submitted

1 Introduction

It has long been known that relying on a single cloud provider is not feasible for certain applications and use-cases [1, 2]. Instead, multiple clusters can be used in a joint effort to offer increased performance. This collaboration between a group of autonomous clusters of computational resources is commonly referred to as a cluster *federation*. Together with emerging infrastructure provider alternatives such as edge computing [3, 4], driven in part by the 5G evolution [5], it is increasingly more important to make autonomous clusters of cloud infrastructures cooperate to jointly provide applications in a more desirable manner. Edge locations are further destined to outnumber the low number of major public cloud providers and regions, thus causing a more pressing need for software to manage underlying systems [6]. To put things into perspective, as of June 2020 Google offers cloud infrastructure in a mere 23 regions. In contrast, edge computing opens the Infrastructure-as-a-Service market to telco providers, and thus dramatically changes the scale and nature of the infrastructure being offered to customers. In Germany alone, and owned by a single telco provider (Deutsche Telekom), some 36,000 base stations will be operational by 2021 [7]. Regardless of whether all or "merely" ten percent of these will offer edge computing facilities, it is clear that methodologies and technologies supporting management and use of such infrastructure will require novel research.

Regarding these new federated environments together with recent trends in shifting application architecture from single monolithic implementations to collections of networked microservices, which in itself adds new complexities on cluster management [8, 9], new questions arise in how to automatically orchestrate applications to provide services of desirable performance. A most interesting and important aspect are the new dimensions that appear to well-known research problems in networking and automatic control, such as load-balancing, auto-scaling and dynamic routing, which is illustrated in Figure 1. For instance, cluster federations are by nature highly dynamic environments, where uncertain cluster-to-cluster connections with potentially non-stationary latencies on top of existing intra-cluster dynamics will make application robustness an issue. Strategies for inter-cluster load-balancing and routing between services, or auto-scaling of individual services will need to adapt to changing dynamics in real-time to guarantee performance. Further, new problems arise in service scheduling and migration when an entire federation of clusters is considered, and where objectives and constraints in client-to-service or service-to-service latency and network bottlenecks might exist and be sporadically violated due to changing dynamics.

Research problems in these fields are commonly tackled using simulation

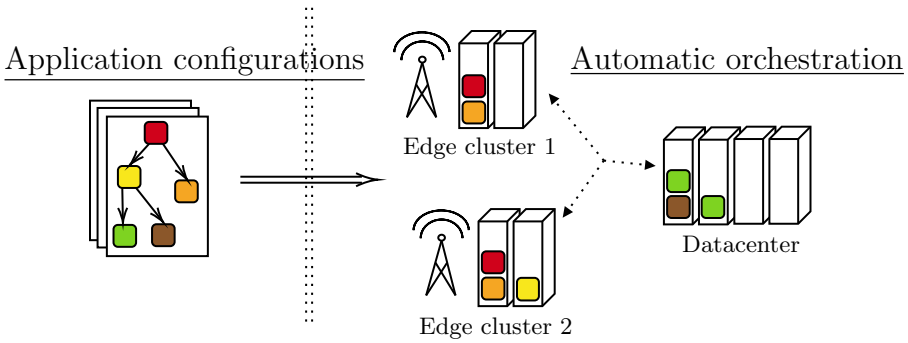


Figure 1: How to automatically decide where to place, route and scale the dependent services constituting an application with associated performance objectives and constraints, under the increased uncertainty and dynamics of a cluster federation is an open problem.

or emulation models [10, 11, 12], which are often quick to set up and play an important part for initial algorithm design and testing but ultimately fails to capture actual system behaviour. To complement this, researchers has the option to perform actual experiments in real environments. However, creating such an environment is in general difficult, expensive and time consuming, and the step required to go beyond simulations to explore ideas in real settings quickly become very large when dealing with complex systems such as cluster federations. To the best of our knowledge, there is currently no easy way for the common researcher to cross this gap and access a desired federation of clusters.

An interesting research question then arises in how a research prototype for federated cloud environments can be designed to best support application orchestration research in networking and automatic control. To this end, we have created the sandbox FedApp with the goal of providing such an environment that (a) is flexible, yet remains a close approximation of real systems, (b) is easy to both deploy and use, and (c) has an easily extendable implementation. Its key features include:

1. creation of a user-defined, multi-cluster virtual environment on OpenStack, complete with Kubernetes and Istio for easy deployment of federation-wide applications.

2. possibility of inducing network characteristics such as delay and loss between clusters, thus enabling faithful emulation of real-world federations.

3. a centralized structure complete with tools for controlling and monitoring the entire federation, supplying the means for which to develop new automatic control and networking solutions.

Upon publication, the sandbox will be released as open source. It is included as a supplementary file in the review process.

The rest of the paper is organized as follows; In Section 2 the tools used to construct our sandbox is presented. In Section 3 we later go into detail on how the sandbox is designed and how it relates to our goal. In Section 4 we then present how the sandbox can enable research in the areas of networking and automatic control in federated application orchestration. Finally, in Section 5 we present a proof-of-concept edge computing scenario to showcase the sandbox.

2 Building a federation

There today exist a myriad of concepts and software tools to implement the functionalities necessary for multi-cluster setups to deliver what could be considered a federated behaviour. In this section we go through the major tools and concepts that lay the foundation for creating the federated environment within our sandbox, and give motivations to why we chose to rely on these.

2.1 OpenStack; Infrastructure virtualization

The sandbox will need some type of infrastructure to be deployed upon. To provide this in a both flexible and easy to use manner, we chose to supply the means of creating a suitable virtual infrastructure via OpenStack. OpenStack is an open source software stack for creating clouds on bare-metal infrastructures. It is commonly used to setup private clouds, where access to underlying hardware is granted, and exclusive access guards against “noisy neighbors” from public clouds [13].

The virtual infrastructure provides the means of creating a controlled environment in which the rest of the sandbox can be deployed in a standardized manned. Supplying a means to create the infrastructure further makes the sandbox flexible, as there is a standardized way of scaling the size, and amount of clusters. On the contrary, making users provide their own bare-metal backends or relying on test-beds such as BonFIRE [14] does not have these benefits.

2.2 Kubernetes; Application management

Cloud workloads are increasingly being packaged and deployed using containerization technologies such as Docker, CRI-O, and containerd. To handle larger containerized deployments, an abstraction layer between raw infrastructure and applications is commonly introduced in the form of a container orchestrator, such as the well-known Kubernetes. Its popularity makes it possible to build Kubernetes-native software which is less tied to any one particular cloud provider, data center or edge cluster, and are therefore able to better make use of the global data center market without concerning users with differences between underlying technology stacks.

In multi-cluster setups, such standardization provides a vital platform for building federation-wide applications that can easily be deployed over a wide array of heterogeneous cluster backends. It is hard to believe that the popularity of Kubernetes will diminish anytime soon, thus providing a multi-cluster environment with Kubernetes allows researchers to experiment in a popular and well-maintained open-source setting which captures real system behaviour. Further, Kubernetes natively supports automatic control strategies such as auto-scaling, which makes great entrypoints for researchers to apply their own solutions.

2.3 Istio; Simplified inter-cluster communication

Referring to a multi-cluster Kubernetes environment as federated is a bit of a stretch, as it is hard to get any non-trivial inter-cluster collaboration to provide joint deployments. To truly be a sandbox for federated clusters, some software is needed to enable cooperative deployments of large scale and complexity. In the sandbox this behaviour is supplied via the tool Istio, a service-mesh software deployed on top of Kubernetes which simplifies steering of network traffic between microservices. What makes it great for this context, is that Istio natively supports creating a single service mesh that span more than one cluster. Thus by launching Istio with replicated control planes across all Kubernetes clusters, a single, federation-wide service mesh is created. As with Kubernetes, the traffic handling via Istio gives an entrypoint for researchers as the service mesh natively performs strategies for e.g. load-balancing and routing. Further, Istio can be used to collect tracing information of service-to-service traffic, which enables measurement of important metrics such as latencies.

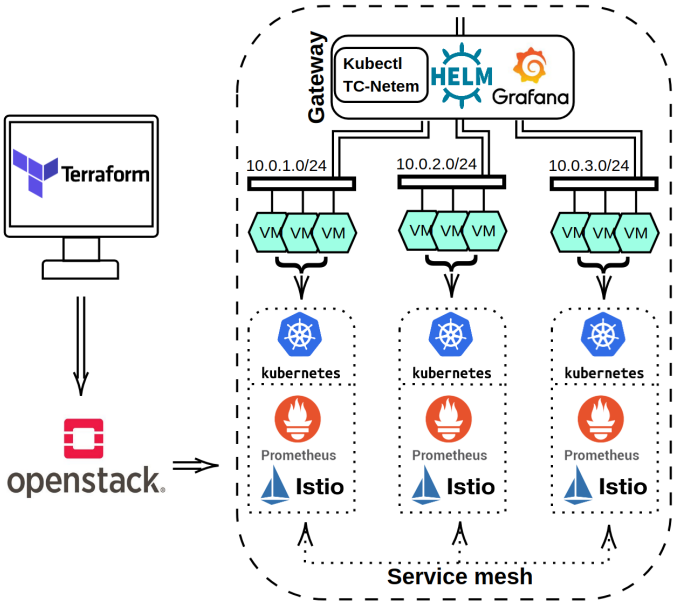


Figure 2: Illustration of the complete sandbox with 3 clusters containing 3 virtual machines each.

3 Sandbox design

Here the design choices for creating the emulated federation to fulfill our goal is presented and described at a detailed conceptual level. We first discuss our choice of network topology that provides the federation emulation, later we explain how we can induce realistic network characteristics, and finally we describe how the sandbox is implemented and how it simplifies the deployment of federation-wide applications on top of it.

An illustration of the complete setup can be seen in Figure 2 which can be used as an overview picture of the sandbox in its entirety. The details of it will be further discussed in Section 3.3.

3.1 Topology of the emulated federation

As we want to emulate a federated cloud setup, each cluster is provisioned in OpenStack as a set of virtual machines (VM) with their own isolated internal network. To enable inter-cluster communication, these internal cluster net-

works need to be connected in some manner. Further, in order to be a close approximation to a real cluster federation, it is vital that users are able to impose desired network characteristics between clusters. To enable this we in this sandbox chose an approach based on using a centralized gateway, where each internal network for a cluster is connected to a gateway virtual machine, which enables us to use the Gateway VM as a single point handler of routing and network emulation between clusters.

Given the stated goal of this project, centralizing the network through a VM in this manner is highly beneficial. First, it enables researchers to affect network characteristics and observe network behaviour such as traffic patterns and workload classifications from a single point. Moreover, a centralized VM increases usability as it can be shipped with all the tools necessary for controlling, monitoring and deploying applications on the clusters, e.g. by giving a single point to inject smart cross-cluster load-balancing features to a federation-wide application and observing the behaviour. Finally, the choice of pooling all functionalities for commandeering the emulated federation makes the setup easy to extend, and easy to debug when something eventually breaks down.

From a performance and scalability perspective, a centralized connection constitutes a bottleneck. However, scalability to support hundreds or more clusters is not the goal in this iteration of our work. Interesting phenomena to study in networking and automatic control, discussed in Section 4, does not need such large realizations to occur. Further it is easy to believe that for most researchers, access to such OpenStack instances where one could allocate that many complete clusters is limited. For this particular sandbox, we thus found it more important to focus on usability rather than scalability. Readers familiar with OpenStack will note that to reverse this design decision and achieve scalability to the level that OpenStack can provide, a single OpenStack Router can be added instead of our centralized Gateway VM. The networks of the different clusters would then all be attached to this OpenStack Router and routing would happen through that router instead. Doing so, however, would make it more difficult to carry out the research we aim to support in this iteration of our software.

3.2 Network characteristics emulation

Benefiting from the chosen centralized topology, the procedure to mimic a real-world networking environment is greatly simplified. Each cluster sees itself as a “stand-alone” cluster on a private network, where all the inter-cluster communication traffic is handled by the Gateway VM. Thus by only affecting the inter-cluster routing logic on the gateway, an arbitrary inter-cluster network profile could be emulated.

We achieve this by fully utilizing the Linux Traffic Control (TC) utility on the gateway. Together with the TC network emulator (TC-netem)[5], it provides the possibility of adding packet loss, delay and other characteristics on the packets from a selected network interface controller (NIC). By applying desired emulated characteristics on the NICs of the Gateway VM, the clusters are not aware of the network status in beforehand. Thereby, the network characteristic to a cluster can be specified arbitrarily, mimicking the propagation and transmission delay caused by the geographic distance to e.g. a edge cluster or data center.

By deploying TC classful queuing discipline (qdisc) and filter, it is further possible to also control the point-to-point network characteristics between clusters. On each interface of the Gateway VM to its connected cluster, all traffic flows are classified and filtered according to the source addresses (the address of another cluster). Each class can then be given its individual TC-netem configuration for a desired cluster-to-cluster network characteristic. This way, only traffic of point-to-point communication between clusters is affected, important meta-communication to the Gateway VM such as cluster monitoring data or control commands are unaffected.

Using TC and TC-netem in this manner on the gateway makes it possible to emulate different network characteristics or scenarios by e.g. (1) applying desired network characteristics on all traffic on a gateway-cluster connection, (2) emulating service or data center/edge cluster outage by applying 100% loss on all traffic on a gateway-cluster connection, or (3) adding point-to-point inter-cluster network characteristics by e.g. reading a matrix map where indices D_{ij} sets the characteristic between clusters i and j.

3.3 Implementation details

As we set out to make the sandbox flexible, easy to use and extendable, we will in this subsection explain on a conceptual level the implementation choices made in its construction to fulfil these goals. To run and deploy the sandbox, researchers only need access to an OpenStack cluster and a computer with Terraform and Ansible installed. The deployment itself is heavily scripted and requires little input from the user part from changing in the settings files and running the script associated with the deployment steps shown in Figure 3.

The sandbox deployment is performed in two distinct stages. In the first and quick stage Terraform, a tool capable of managing the life-cycle of resources at various services such as OpenStack, is used to deploy the Gateway VM and Ansible used to install dependencies. From the gateway, the rest of

[5] `https://www.linux.org/docs/man8/tc-netem.html`

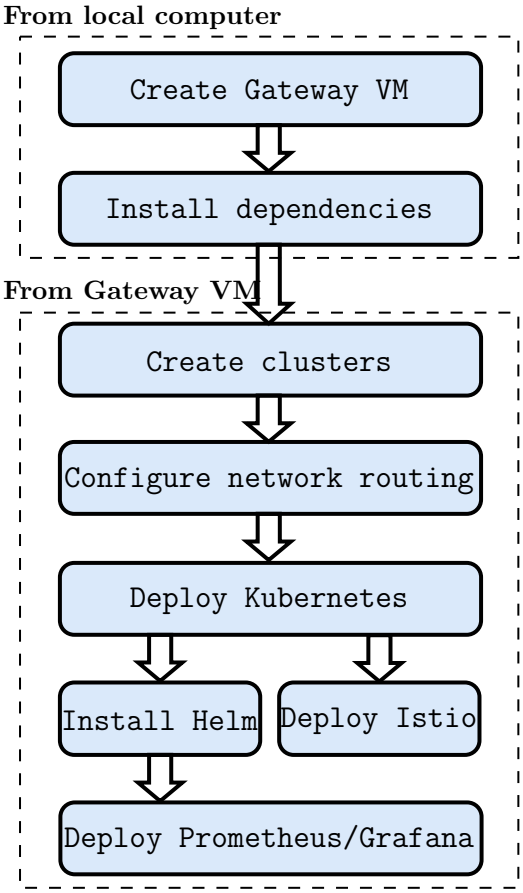


Figure 3: Illustration of the different steps in the sandbox deployment. Each box represents a single script or tool to be run, with a corresponding set of configuration files.

the sandbox deployment can then be performed in the cloud. There are two major advantages to this. First, the sandbox in its entirety might take quite some time to deploy. Secondly, tools such as Terraform exports configuration files on completion, which can then be directly imported into other tools on the gateway. From the Gateway, a Python script is then used to call Terraform repeatedly to provision the user defined virtual infrastructure for each cluster. To provide means of easy routing between clusters, each internal cluster net-

work is assigned a sandbox-unique CIDR. After Terraform completes, routing tables on the gateway are configured and the aforementioned TC-netem setup to enable the inter-cluster network emulation.

Once the virtual infrastructure has been provisioned, Kubespray is leveraged via a Python script and pre-supplied configuration files to deploy individual Kubernetes instances on all clusters. Kubespray is a tool that provides simplified setups of industry-standard and well-configured Kubernetes clusters. On completion, the authentication for API calls to each cluster is added as a context to a single instance of Kubectl at the Gateway VM. Kubectl is Kubernetes own command tool, and it enables each cluster in the federation to be directly controlled from the gateway by a simple switch between contexts. To simplify deployment of federation-wide applications, Helm, a commonly used tool to deploy complex applications to a Kubernetes cluster, is then installed on the Gateway and the Kubernetes clusters updated to support it. On top of Kubernetes, Istio is then deployed to each cluster with a replicated control plane to create the federation-wide service mesh. Finally, to provide insight on the behaviour of the federation, it is vital that cluster nodes and applications are monitored. To this end the sandbox comes supplied with federation-wide monitoring in the form of data-collection using Prometheus, and a Grafana GUI deployed on the Gateway VM. Via a Python script, Helm is utilized to deploy the Prometheus to each cluster and Grafana in a docker container on the gateway with the correct Prometheus backends.

The deployment capitalizes on the natural modularity between the different tools, and there is no master script or single configuration file to run the entire deployment at once. Instead the deployment is broken down to scripts and configuration files that handles each distinct step separately. The modularity makes the sandbox highly flexible and extendable as it easy to change in one part of the deployment without it affecting the other parts of the sandbox. It is further trivial to add completely new parts to the sandbox setup. Further, if one part of the deployment breaks, the modularity makes the sandbox much easier to debug. Despite the modularity the entire deployment is still heavily scripted, to the point that each part can be completed with running a single script with corresponding configuration files. The scripting is performed such that each stage after allocating the virtual infrastructure is invariant to the virtual infrastructure configurations, such as the number of clusters or VM's within those clusters. An exception is the instance image, but if one uses a standard Ubuntu 18.04 for all VMs this should pose no trouble, and the pre-supplied configuration files should in this case enable a sandbox almost deployable out-of-the-box.

The inclusion of Helm and Istio further makes it possible to create a streamlined method to deploy complex federation-wide applications to the sandbox

directly from the Gateway VM. Given an application whose services are packaged into docker images, and uploaded to a container registry reachable from the Kubernetes clusters, each cluster microservice deployment can be specified with its own set of Helm configuration files (known as a Helm Chart). These Charts keep track of the different internal microservice logics, such as to which microservice Istio should route what response and how the clusters should access the container registry. The exact clusters to deploy what microservice to, or the IP address of a microservices where to route a response does not need to be known in the Charts, only the routing logic on a microservice-to-microservice basis. The actual clusters and IPs can then instead be handled by an external Python scripting file, that via Kubectl extracts and inputs the correct values to the Charts before using Helm to deploy the microservices. This amounts to a general and organized way of deploying federation-wide applications on the sandbox whose microservices are independent of which clusters they are deployed to. Where to deploy what and by what quantity can simply be defined in a single line in the Python script, and adding a new microservice (once the source code of the application has been updated) becomes the simple task adding a new Helm Chart and possibly changing the Helm Charts of its in- and out-connected microservices. In the sandbox, the included example application presented in Section 5 utilizes this pipeline for deployment, which can easily be adapted to fit other applications.

4 Functionalities

The sandbox was created with the goal of providing a federated cloud environment for application orchestration research in the fields of networking and automatic control. In this section we discuss its functionalities that could specifically contribute to this point. Via our design, researchers can set up an user-defined federated Kubernetes environment, with a single centralized point for collecting all tools and scripts needed for researchers to deploy, monitor and control federation-wide applications. Because of the centralized structure and our emphasise on extendability, new tools or software that are not supplied out-of-the-box can easily be incorporated into the sandbox. The decision for provisioning the infrastructure with OpenStack together with TC-netem for network emulation further enables researchers to experiment in a wide range of federated cloud or edge computing settings.

4.1 For control-oriented research

To enable research in automatic control there need to be two functionalities readily available, namely methods for measurement and actuation. To give an example, consider horizontal auto-scaling where it is desired to keep average CPU utilization in some interval. Measurement implies monitoring CPU utilization of the replicas, while actuation is the mean of which we can influence behaviour, i.e. by adding or removing replicas. The control algorithm is then the logic which dictates when and how to actuate, in response to the measurements in order to keep the CPU utilization within the given interval.

Considering measurements, it is possible from the tools of the sandbox to collect a large array of different and useful data. The metrics gathered via Prometheus include information such as CPU and memory usage for services and nodes, which are necessary for control algorithms that can react to e.g. resource shortages or over-provisioning. Further, via Istio one could extract tracing data that contains information about latencies for services, which is needed for control algorithms that tries to e.g. keep request response time percentiles below a certain level. Not only can this data be accessed for control within a single Kubernetes cluster, but it can be fetched from other clusters, enabling researchers to explore control algorithms that take into account the behaviour of microservices downstream in the federation-wide activation chain.

Considering actuation instead, both Kubernetes and Istio natively support auto-scaling and load-balancing, which gives researchers a great entrypoint to implement their own solutions. Further, the centralized Gateway VM enables control of all clusters in the federation via the single Kubectl instance, and could be used design control algorithms for things such as microservice migration between clusters.

The pressing control questions to tackle lies in the need to perform control from a more holistic overview of the entire federation-wide application in order to achieve higher level performance goals. Examples include keeping user response times low while minimizing total resource usage, providing fail-overs without impact to total service performance, or automatically discovering and avoiding pitfalls from back-pressure or tier dependencies between microservices, all with minimal DevOps interaction. For this we hope that the sandbox can be of use, by enabling the development of smart control algorithms that can utilize metrics and tracing data from the entire federation, and exert control via multiple actuators.

4.2 For network-oriented research

Network related research is not orthogonal to research in control, much of the functionalities of measurement and actuation from the previous section can be applied here as well. In particular, Istio enables full control and observation of application traffic within a cluster. It further allows the control of inter-cluster traffic for federation-wide applications. Unfortunately traffic is monitored independently in each cluster without collaboration. Here our centralized structure is beneficial, as all traffic passes through it, observation of all inter-cluster traffic and interactions can be performed here.

Istio further makes it easy to create, control and observe complex federation-wide microservice networks. Together with the ability to apply arbitrary network characteristics via TC-netem between sites, the sandbox offers an easy way to implement and test different network architectures under varying scenarios, enabling researchers to evaluate solutions for improving service performances. For instance a small city-wide mobile edge cloud network could be emulated in the sandbox, with each cluster representing an edge site, with the purpose of developing and testing solutions for resource allocation and service migration algorithms.

The network emulator further provides an opportunity to explore the impact of network characteristics such as delay and loss on services. It can help in the development of strategies for delay constrained networking systems in order to provide better performances in traffic management and scheduling. Likewise, it can be used to study the fault-tolerance of the different strategies under specific scenarios.

Finally, the sidecar-proxy system that Istio uses to link microservices in order to create the service mesh makes it possible to enforce application-independent encryption on links. Together with the ability of apply different security rules for clusters in accordance with the functionalities of services they host, makes it possible to use the sandbox to study and analyse cloud application security and privacy in a federated setting.

5 Proof-of-concept

To showcase some of the functionalities, we here present an edge computing scenario with a federated example application deployed on top of the sandbox. In the scenario we will consider a face detection application that in order to save bandwidth, has had its image pre-processing off-loaded to two smaller edge clusters C_1, C_2 from the actual classification at two larger data centers C_3, C_4.

5.1 Deploying the scenario

As explained in Section 3.3, once a user has access to an OpenStack cluster, the implementation choices of the sandbox makes the deployment straightforward by following the steps staked out in Figure 3. Before deploying, the only major input required is the two Terraform configuration files tied to the allocation of the Gateway VM and the four clusters, which needs to be updated with the desired values. Without loss of generality, each virtual edge cluster and data center were given the same size of four VMs each with four vCPUs and 8 Gbytes of RAM. The Gateway VM was further deployed with four vCPUs and 16 Gbytes of RAM. All of the VMs were given Ubuntu 18.04 as the instance image.

In order to emulate geographic distance, a delay was introduced using TC-netem between the edge clusters C_1, C_2 and the backend cluster C_3 with the following delay matrix.

$$D = \begin{pmatrix} 0 & 0 & 25 & 0 \\ 0 & 0 & 25 & 0 \\ 25 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \; ms$$

As explained in Section 3.2, D_{ij} tells TC-netem to introduce a one way delay of the given amount between clusters C_i and C_j. This means that the roundtrip-time (RTT) between C_1, C_2 and C_3 should be 50 ms.

For the example application, frontend image pre-processing and backend classification where delivered in two standalone microservices, packed into docker containers and uploaded to a container registry on GitLab. The frontend microservice reads a RGB image from an API call and reduces it to a grayscale image. The smaller grayscale image is sent to the backend microservice, where the detection is performed using a standard Haar-cascade classifier in OpenCV. If a face is detected, the corners of the bounding box is returned to the frontend, which adds the box to the original RGB image and sends it back as a response to the original API call. Using the streamlined method discussed in Section 3.3, the frontend microservices were deployed with two replicas on each edge cluster, while the backend microservices used three replicas on each data center. Further, in order to balance requests between the multiple edge clusters, an NGINX load balancer was launched in a docker container on the Gateway VM. An overview of the scenario setup can be seen in Figure 4.

In order to put any real stress on the system, a load generator was further created which sends images at a given rate to the NGINX load balancer at the Gateway VM. For this demonstrative scenario, the necessary images where provided by the UMass face detection data set and benchmark[15].

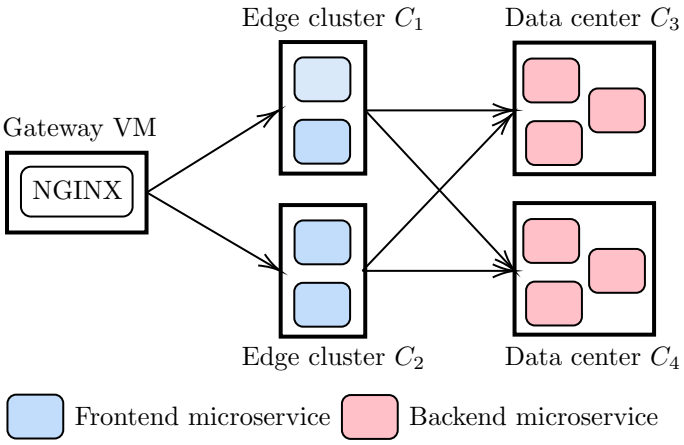


Figure 4: Emulated edge clusters and data centers hosting the two frontend and three backend microservices for the example federated face detection application.

5.2 Experimental evaluation

To see whether the system behaves as expected, a handful of experiments were performed on the example scenario. Initially, we examined how well TC-netem managed to create our desired network characteristics. This was done by measuring the RTT by running `ping` between every cluster simultaneously with an inter-sample time of 0.2 s over 600 s. We then examined if the cluster software stacks managed to properly handle the requests despite various disturbances, by observing the application vCPU usage at the different clusters when putting stress on the system. Thanks to the federation-wide monitoring, metrics such as vCPU for all clusters is easily observed in Grafana.

The impact of two different inter-cluster load balancers in Istio, Round Robin (RR) and Least Connection (LC), under different loads was first examined. The RR load balancer distributes the requests evenly amongst the backends, which makes it largely unaffected by delay, while LC on the other hand prioritizes sending requests to the backend which has the least amount of active requests, making it sensitive to our added delay. Images were sent at both 10 images/s and 20 images/s for a time of 600 seconds per rate. If all is working properly, we would expect to see a vCPU usage lower for the edge clusters C_1, C_2 than for the data centers C_3, C_4 as these perform a more computationally heavy task. We further expect to see a doubling of vCPU usage when the load generation rate is doubled. Further, we expect to see roughly

equal vCPU usage on C_3, C_4 using the RR policy, while using LC policy would prioritize C_4 without delay.

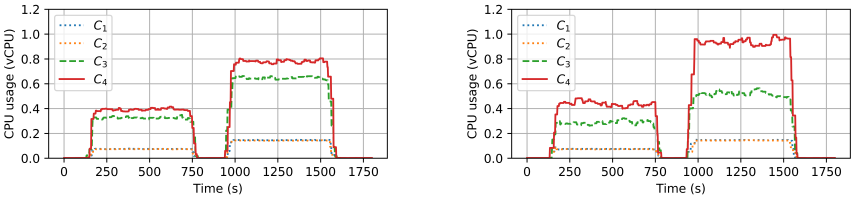
We then examined how well the emulation could reproduce sudden cluster link failure. The introduced delay was removed and RR load-balancing used for the inter-cluster load-balancing. The load generator was used to send 20 images/s to the system for 1200 s. At 300 s and 900 s the connection to cluster C_3 is suddenly lost for 300 s, by introducing a 100% loss rate on the link of C_3 using TC-netem. We would in this case expect that the vCPU usage of C_3 reduce to 0, and the vCPU usage of C_4 to double during the outages.

For the RTT experiment, the resulting means and variances of the measured RTT for every link was

$$\mathbb{E}\left[2\widehat{D}\right] \approx \begin{bmatrix} 0.05 & 1.48 & 51.7 & 1.56 \\ 1.46 & 0.04 & 51.8 & 1.46 \\ 51.7 & 51.8 & 0.05 & 1.59 \\ 1.47 & 1.45 & 1.58 & 0.03 \end{bmatrix} \; ms$$

$$\mathbb{V}\left[2\widehat{D}\right] \approx \begin{bmatrix} 0.01 & 0.02 & 0.11 & 0.03 \\ 0.03 & 0.00 & 0.05 & 0.02 \\ 0.04 & 0.09 & 0.00 & 0.10 \\ 0.02 & 0.02 & 0.05 & 0.00 \end{bmatrix} \; ms$$

As can be seen, the results corresponds well to the desired values. A small bias can be observed due to the internal OpenStack network, but the variance remains low for all links. For our load generation experiments, the results can be seen in Figure 5 and Figure 6. For all graphs, the results looks as expected indicating that the sandbox works as intended. The slight difference in RR CPU usage could be contributed to the underlying VM allocations. As the measurement constitutes the rate of change in container CPU usage in seconds, a faster clock speed would lead to lower CPU usage. We can however be fairly certain that the RR difference is not due to the added delay, as the results in Figure 6 which has no delay experiences similar differences as the results displayed in Figure 5a. Further, the apparent noisiness could be attributed to the different sizes of images in the dataset in conjunction with the inherent uncertainty in execution speed on the cloud. To let researchers try out this example by themselves, and to demo how a federation-wide application can be implemented and deployed in FedApp, the example application is included in the sandbox.



(a) Using Istio's RR policy for inter-cluster load-balancing.

(b) Using Istio's LC policy for inter-cluster load-balancing.

Figure 5: CPU usage of the frontends on clusters 1 and 2, and the backends on clusters 3 and 4 under the two different load generation rates of 10 images/s and 20 images/s. The two graphs displays different inter-cluster load-balancing policies.

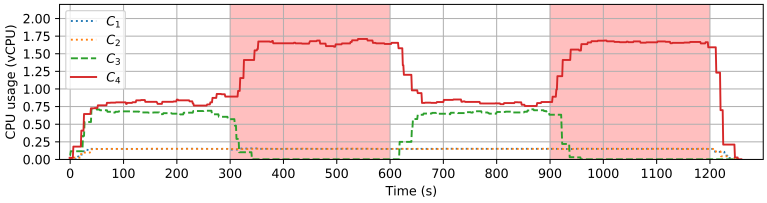


Figure 6: CPU usage of the frontends on clusters 1 and 2, and the backends on clusters 3 and 4 under a load generation rate of 20 images/s. Shaded red areas show when the connection to backend cluster 3 has been lost.

6 Related work

In older literature, the matter of multi-cluster collaboration was focused on bridging technological gaps between competing cloud infrastructure providers with possibly heterogeneous underlying technology stacks [1]. The vision of inter-cloud compatibility was embodied by shared or compatible APIs on the infrastructure level [16, 17, 18]. With more than a decade of hindsight, it is clear that these early attempts failed to convincingly demand industry adoption among the major cloud providers.

Recent work shows that a Kubernetes-based edge deployment is promising for real-life applications, and offers improvements with regard to e.g. latency [19, 20, 21]. In addition to relying on Kubernetes to provide a common

platform abstraction, cloud-native applications can leverage service meshes such as Istio to simplify networking code [22].

Compared to other approaches that aims to enable research for cloud systems, the sandbox differentiates itself in a number of ways. Its flexibility and openness puts it in stark contrast to other conceptually similar approaches, such as [23]. Further, in regards to the numerous Fog/Edge emulators, see Section 4.14.1 in [24], cloud simulators such as CloudSim [25], or direct modeling of Kubernetes-based platforms, such as [26, 27], the sandbox enables researchers to utilize the entire cluster software stack. We believe that due to the rapid pace of development in cloud orchestration tools such as Kubernetes and Istio, a practical sandbox that lets researchers deploy the actual software and not models thereof, have a greater impact and thus provide more valuable future insights.

7 Discussion

The way the cluster federation in our sandbox is constructed is motivated from our focus to support research in networking and automatic control of applications in these settings. It is worth mentioning that these implementation choices are not the only way to create such an environment, especially with another focus in mind a different approach with different tools might be more suitable. For example, if one desires to test out e.g. application scaling performance over a very large amount of clusters, then the Gateway VM bottleneck will quite likely have a negative impact.

Moreover, the current implementation seems to have a maximum number of possible network interfaces for an OpenStack VM that would need to be circumvented for such a scenario. On our specific OpenStack instance we managed to allocate up to 22 clusters before running into this interface limit on our Gateway VM. A different OpenStack implementation or instance image might have a different interface limit. For these 22 clusters, we however experienced only negligible effects on network performance in a low traffic scenario, which hints that the sandbox can handle much larger experiments than our demonstrative scenario in Section 5. Ultimately, it would have been a nice feature to support an even larger amount of clusters but as explained in Section 3.1 it is simply not the type of research that we have aimed to support in the current iteration of our work.

Besides our aims, there are other use-cases which the sandbox could support, or extended to support in its current form. For example, FedApp has the potential to deliver an environment to test out performance of specific federation software and tools. Kubernetes own federated deployment, KubeFed v2

which is currently in its alpha comes to mind.

Looking forward, apart from our simple example it would be of benefit to out-of-the-box include other open source benchmark applications such as the ones introduced by Gan et al. [9]. Further, implementing and deploying algorithms in Kubernetes or Istio is not a trivial matter for the inexperienced researcher. A benefit to usability would be to streamline this process as well, either via providing concrete examples or by some toolkit.

8 Conclusion

In this paper we have presented FedApp, a tool for providing a federated Kubernetes environment in OpenStack with user-defined network characteristics between clusters. The goal has been to give researchers in the fields of networking and automatic control a way to quickly set up federated environments for experimentation. Upon publication, the sandbox will be fully available as open source via GitHub.

Acknowledgement

This work is partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Also, the work in this paper is partially supported by the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the 5G-PERFECTA Celtic Next project funded by Sweden's Innovation Agency (VINNOVA). Johan Ruuskanen & Maria Kihl is part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT).

References

[1] B. Rochwerger, A. Galis, E. Levy, J. A. Caceres, D. Breitgand, Y. Wolfsthal, I. M. Llorente, M. Wusthoff, R. S. Montero, and E. Elmroth. RESERVOIR: Management technologies and requirements for next generation service oriented infrastructures. In *2009 IFIP/IEEE International Symposium on Integrated Network Management*, 2009. doi: 10.1109/INM.2009.5188828.

[2] Ana Juan Ferrer, Francisco Hernández, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Raül Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Srijith K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forgó, Tabassum Sharif, and Craig Sheridan. OPTIMIS: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012. doi: https://doi.org/10.1016/j.future.2011.05.022.

[3] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5): 637–646, 2016.

[4] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, Nov 2016. ISSN 2372-2568. doi: 10.1109/MCC.2016.124.

[5] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[6] Ronan-Alexandre Cherrueau, Adrien Lebre, Dimitri Pertin, Fetahi Wuhib, and João Monteiro Soares. Edge computing resource management system: a critical building block! initiating the debate via openstack. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, July 2018. USENIX Association. URL `https://www.usenix.org/conference/hotedge18/presentation/cherrueau`.

[7] Pascal Kiel-Koslowski. Deutsche telekom commissions over 300 new LTE mobile base stations. URL `https://www.telekom.com/en/media/media-information/archive/300-new-lte-mobile-base-stations-574106`.

[8] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari. Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing*, 3 (5):81–88, Sep. 2016. ISSN 2372-2568. doi: 10.1109/MCC.2016.112.

[9] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, and et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 3–18, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi: 10.1145/3297858.3304013. URL `https://doi.org/10.1145/3297858.3304013`.

[10] A. Ahmed and A. S. Sabyasachi. Cloud computing simulators: A detailed survey and future direction. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 866–872, Feb 2014. doi: 10.1109/IAdCC.2014.6779436.

[11] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. A comparative analysis of simulators for the cloud to fog continuum. *Simulation Modelling Practice and Theory*, page 102029, November 2019. doi: 10.1016/j.simpat.2019.102029. URL `https://doi.org/10.1016/j.simpat.2019.102029`.

[12] B. Ramprasad, M. Fokaefs, J. Mukherjee, and M. Litoiu. Emu-iot - a virtual internet of things lab. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*, pages 73–83, 2019.

[13] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, June 2011. ISSN 2161-9883. doi: 10.1109/TPDS.2011.66.

[14] Alastair C. Hume, Yahya Al-Hazmi, Bartosz Belter, Konrad Campowsky, Luis M. Carril, Gino Carrozzo, Vegard Engen, David García-Pérez, Jordi Jofre Ponsatí, Roland Kűbert, Yongzheng Liang, Cyril Rohr, and Gregory Van Seghbroeck. BonFIRE: A multi-cloud test facility for internet of services experimentation. In *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 81–96. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-35576-9_11. URL `https://doi.org/10.1007/978-3-642-35576-9_11`.

[15] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[16] E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *2009 Eighth International Conference on Grid and Cooperative Computing*, pages 253–260, Aug 2009. doi: 10.1109/GCC.2009.36.

[17] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an open cloud standard. *IEEE Internet Computing*, 16(4):15–25, 2012.

[18] Boris Parák, Zdenek Sustr, Florian Feldhaus, Piotr Kasprzak, and Maik Srba. The rocci project: providing cloud interoperability with occi 1.1. In *International Symposium on Grids and Clouds (ISGC) 2014*, volume 210, page 014. SISSA Medialab, 2014.

[19] Franco Cicirelli, Antonio Guerrieri, Giandomenico Spezzano, and Andrea Vinci. An edge-based platform for dynamic smart city applications. *Future Generation Computer Systems*, 76:106 – 118, 2017. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.05.034. URL `http://www.sciencedirect.com/science/article/pii/S0167739X16308342`.

[20] P. Tsai, H. Hong, A. Cheng, and C. Hsu. Distributed analytics in fog computing platforms using tensorflow and kubernetes. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 145–150, Sep. 2017. doi: 10.1109/APNOMS.2017.8094194.

[21] Yuzhou Huang, Kaiyu cai, Ran Zong, and Yugang Mao. Design and implementation of an edge computing platform architecture using docker and kubernetes for machine learning. In *Proceedings of the 3rd International Conference on High Performance Compilation, Computing and Communications*, HP3C '19, page 29–32, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366380. doi: 10.1145/3318265.3318288. URL `https://doi.org/10.1145/3318265.3318288`.

[22] Ozair Sheikh, Serjik Dikaleh, Dharmesh Mistry, Darren Pape, and Chris Felix. Modernize digital applications with microservices management using the istio service mesh. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, CASCON '18, page 359–360, USA, 2018. IBM Corp.

[23] Endah Kristiani, Chao-Tung Yang, Yuan Ting Wang, and Chin-Yin Huang. Implementation of an edge computing architecture using openstack and kubernetes. In Kuinam J. Kim and Nakhoon Baek, editors, *Information Science and Applications 2018*, pages 675–685, Singapore, 2019. Springer Singapore. ISBN 978-981-13-1056-0.

[24] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, September 2019. doi: 10.1016/j.sysarc.2019.02.009. URL `https://doi.org/10.1016/j.sysarc.2019.02.009`.

[25] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, August 2010. doi: 10.1002/spe.995. URL `https://doi.org/10.1002/spe.995`.

[26] V. Medel, O. Rana, J. Á. Bañares, and U. Arronategui. Adaptive application scheduling under interference in kubernetes. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 426–427, Dec 2016.

[27] Víctor Medel, Rafael Tolosana-Calasanz, José Ángel Bañares, Unai Arronategui, and Omer F. Rana. Characterising resource management performance in kubernetes. *Computers & Electrical Engineering*, 68:286 – 297, 2018. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2018.03.041. URL `http://www.sciencedirect.com/science/article/pii/S0045790617315240`.