Private 5G: A Systems Approach

Release Version 1.1-dev

Peterson, Sunay, and Davie

Oct 18, 2023

TABLE OF CONTENTS

FOREWORD

When I was a graduate student studying parallel computing in the early 1990s, the World Wide Web exploded onto the scene, and I wanted to understand how the Internet made such rapid advances possible. I picked up a copy of the first edition of *Computer Networks: A Systems Approach*, and I started reading. I was delighted to learn not only the Internet's history and main concepts, but also to see examples of real code that illustrated how the protocols actually worked and how they could be changed. I was hooked. I loved the idea of working in the computer networking field, where the technologies we build can help bring people together and lower the barriers to innovation so that anyone who can write software can contribute.

As time went on, I saw that, while the Internet enabled rapid advances in end-host applications, the inside of the Internet was harder to change. Network devices were closed and proprietary, and protocol standards evolved slowly and painfully. In response, enabling innovation inside the network became a passion of mine, and of many other technologists eager to make the Internet better–more secure, performant, reliable, cost-effective, and easier to manage. Gradually, the computer networking field changed, with the advent of software-defined networking, giving network owners–such as the hyperscalers running large data centers–much more control over the software inside their networks. (See *Software-Defined Networks: A Systems Approach* for more of this story!) Nowadays, computer networking really is about software. It's a welcome change.

In recent years, the excitement has moved to the network edge with the emergence of 5G cellular access networks. No longer are wireless communication, computer networking, and cloud computing siloed parts of some larger ecosystem. They are brought together, allowing wireless devices to connect to nearby computing resources. Far beyond providing mobile Internet access, these networks enable exciting real-time applications like augmented and virtual reality (AR/VR), Internet of Things (IoT), self-driving cars, drones, robotic control, and more. Plus, these networks are not only the purview of large carriers that must invest significant resources in wireless spectrum, cell towers, and more. Rather, individual enterprises, campuses, and communities are deploying private 5G within their own organizations to support their own applications, using lightly regulated spectrum like CBRS (Citizens Broadband Radio Service). Opportunities for innovation abound!

However, now there is so much more to learn, to understand the many parts of 5G access networks as a single coherent system. The learning curve can be steep. Even fairly technical people usually know one or at most two parts of the system, and do not know how the parts relate to the larger whole. For example, I knew a good amount about computer networking, a little about cloud computing, and not much about wireless communication. *Private 5G: A Systems Approach*–this book–changed that. I learned what I needed to know about the radio access network, and how the pieces come together to be more than the sum of their parts. More than that, the book shows that, despite the complexity of the 3GPP cellular standards and the long and frustrating history of closed network equipment in the cellular networking space, open source software is gaining a foothold. Open source platforms like Aether and Magma–both discussed in this book–are seeing practical deployment in a wide variety of settings. The book even has a guide for readers to bring up the Aether platform, including a 5G small cell.

For me, then, the story comes full circle. Private 5G networks are something you can touch, code, and deploy yourself. Armed with the knowledge of how 5G access networks work, and with hands-on experience with open source software, just imagine the places you'll go!

Jennifer Rexford
Princeton, New Jersey

PREFACE

When we wrote our introductory 5G book three years ago, our goal was to help people with experience building Internet and cloud services to understand the opportunity to bring best practices from those systems to the mobile cellular network. On paper (and in the press) 5G had set an ambitious goal of transformative changes, adopting a cloud-inspired architecture and supporting a new set of innovative services. But the gap between that aspirational story and the reality of 40 years of network operators and hardware vendors protecting their incumbent advantages made for a challenging pivot. So we started with the basics, and set out to explain the fundamental networking concepts and design principles behind the myriad of acronyms that dominate mobile cellular networking.

Because 5G adopts many of the principles of cloud native systems, it promises to bring the feature velocity of the cloud to Telco environments. That promise is being delivered most successfully in private 5G deployments that are less constrained by existing Telco organizations and legacy infrastructure. What started out as sketches on a whiteboard three years ago is now becoming a reality: Several cloud providers are offering private 5G solutions for enterprises, and there is a complete open source implementation of a 5G-enabled edge cloud that the Internet community can learn from and build upon.

The architecture described in this book is not limited to private deployments. It includes the necessary background information about the mobile cellular network, much of which is rooted in its origin story as a Telco voice network, but the overarching theme is to describe the network through the lens of private deployments of 5G connectivity as a managed cloud service. This includes adopting best practices in horizontally scalable microservices, Software-Defined Networking (SDN), and cloud operational practices such as DevOps. These practices are appropriate for traditional operators, cloud providers, and enterprises alike, but it is emerging use cases in private deployments that will benefit first.

The book makes extensive use of open source software—specifically, the Aether and Magma projects—to illustrate how Private 5G can be realized in practice. The availability of open software informs our understanding of what has historically been a proprietary and opaque system. The result complements the low-level engineering documents that are available online (and to which we provide links) with an architectural roadmap for anyone trying to understand all the moving parts, how they fit together, and how they can be operationalized. And once you're done reading the book, we encourage you to jump into the hands-on appendix that walks you through the step-by-step process of deploying that software in your own local computing environment.

Acknowledgements

The software described in this book is due to the hard work of the ONF engineering team, the Magma engineering team, and the open source communities that work with them. Bilal Saleem did the heavy lifting on Aether OnRamp (described in the *About the Software* Appendix), with a special thanks to Ajay Thakur, Andy Bavier, Gabriel Arrobo, and Muhammad Shahbaz for their guidance and feedback.

Thanks to the members of the community who contributed text or corrections to the book, including:

- Edmar Candeia Gurjão
- Muhammad Shahbaz
- Mugahed Izzeldin
- Robert MacDavid
- Simon Leinen
- Tiago Barros
- Gabriel Arrobo
- Ajay Thakur

The picture of a Magma deployment in Chapter 5 was provided by Shaddi Hasan.

Larry Peterson, Oguz Sunay, and Bruce Davie
April 2023

CHAPTER

ONE

CHAPTER 1: INTRODUCTION

Mobile networks, which have a 40-year history that parallels the Internet's, have undergone significant change. The first two generations supported voice and then text, with 3G defining the transition to broadband access, supporting data rates measured in hundreds of kilobits per second. Today, the industry is transitioning from 4G (with data rates typically measured in the few megabits per second) to 5G, with the promise of a tenfold increase in data rates.

But 5G is about much more than increased bandwidth. 5G represents a fundamental rearchitecting of the access network in a way that leverages several key technology trends and sets it on a path to enable much greater innovation. In the same way that 3G defined the transition from voice to broadband, 5G's promise is primarily about the transition from a single access service (broadband connectivity) to a richer collection of edge services and devices. 5G is expected to provide support for immersive user interfaces (e.g., Augmented Reality, Virtual Reality), mission-critical applications (e.g., public safety, autonomous vehicles), and the Internet of Things (IoT). Because these use cases will include everything from home appliances to industrial robots to self-driving cars, 5G will support not only humans accessing the Internet from their smartphones, but also swarms of autonomous devices working together on their behalf.

There is more to supporting these services than just improving bandwidth or latency to individual users. As we will see, a fundamentally different edge network architecture is required. The requirements for this architecture are ambitious, and can be illustrated by three classes of capabilities:

- To support *Massive Internet of Things*, potentially including devices with ultra-low energy (10+ years of battery life), ultra-low complexity (10s of bits per second), and ultra-high density (1 million nodes per square kilometer).
- To support *Mission-Critical Control*, potentially including ultra-high availability (greater than 99.999% or "five nines"), ultra-low latency (as low as 1 ms), and extreme mobility (up to 100 km/h).
- To support *Enhanced Mobile Broadband*, potentially including extreme data rates (multi-Gbps peak, 100+ Mbps sustained) and extreme capacity (10 Tbps of aggregate throughput per square kilometer).

These targets will certainly not be met overnight, but that's in keeping with each generation of the mobile network being a decade-long endeavor.

On top of these quantitative improvements to the capabilities of the access network, 5G is being viewed as a chance for building a platform to support innovation. Whereas prior access networks were generally optimized for known services (such as voice calls and SMS), the Internet has been hugely successful in large part because it supported a wide range of applications that were not even thought of when it was first designed. The 5G network is designed with this same goal: enabling future applications beyond those we fully recognize today. For an example of the grand vision for 5G, see the whitepaper from one of the industry leaders.

Further Reading

Qualcomm Whitepaper. Making 5G NR a Reality. December 2016.

The 5G mobile network, because it is on an evolutionary path and not a point solution, includes standardized specifications, a range of implementation choices, and a long list of aspirational goals. Because this leaves so much room for interpretation, our approach to describing 5G is grounded in three mutually supportive principles. The first is to apply a *systems lens*, which is to say, we explain the sequence of design decisions that lead to a solution rather than fall back on enumerating the overwhelming number of acronyms or individual point technologies as a *fait accompli*. The second is to aggressively disaggregate the system. Building a disaggregated, virtualized, and software-defined 5G access network is the direction the industry is already headed (for good technical and business reasons), but breaking the 5G network down into its elemental components is also the best way to explain how 5G works.

The third principle is to illustrate how 5G can be realized in practice by drawing on specific engineering decisions made in an open source implementation. This implementation leverages best practices in building cloud apps, which is an essential aspect of 5G evolving into a platform for new services. This implementation also targets enterprises that are increasingly deploying 5G locally, and using it to help automate their manufacturing, retail, and business practices—a trend that has been dubbed *Industry 4.0*. Such enterprise-level deployments are known as *Private 5G*, but there is nothing about the technical approach that couldn't be adopted throughout the more traditional "public mobile network" that comes to mind when you think about your cell service today. The only difference is that private deployments are more aggressively embracing the cloud practices that will ultimately distinguish 5G from earlier generations.

Further Reading

K. Schwab. The Fourth Industrial Revolution. World Economic Forum.

What this all means is that there is no simple definition of 5G, any more than there is for the Internet. It is a complex and evolving system, constrained by a set of standards that purposely give all the stakeholders many degrees of freedom. In the chapters that follow, it should be clear from the context whether we are talking about *standards* (what everyone must do to interoperate), *trends* (where the industry seems to be headed), or *implementation choices* (examples to make the discussion more concrete). By adopting a systems perspective throughout, our intent is to describe 5G in a way that helps the reader navigate this rich and rapidly evolving system.

1.1 1.1 Standardization Landscape

As of 3G, the generational designation corresponds to a standard defined by the *3rd Generation Partnership Project (3GPP)*. Even though its name has "3G" in it, the 3GPP continues to define the standards for 4G, 5G, and so on, each of which corresponds to a sequence of releases of the standard. Release 15 is considered the demarcation point between 4G and 5G, with Release 17 having been completed in 2022.

In addition to 3GPP-defined standards, national governments establish how the radio spectrum is used locally. Unlike Wi-Fi, for which there is international agreement that permits anyone to use a channel at either 2.4 or 5 GHz (these are unlicensed bands), governments have auctioned off and licensed exclusive use of various frequency bands to service providers, who in turn sell mobile access service to their subscribers. The use of licensed spectrum brings certain benefits such as greater control over the quality of service delivered, while also imposing costs both in terms of paying for licenses and in the complexity of the systems needed to manage access to the spectrum. We will explore how these costs and benefits play out in subsequent chapters.

There is also a shared-license band at 3.5 GHz, called *Citizens Broadband Radio Service (CBRS)*, set aside in North America for cellular use. Similar spectrum is being set aside in other countries. The CBRS band allows 3 tiers of users to share the spectrum: first right of use goes to the original owners of this spectrum (naval radars and satellite ground stations); followed by priority users who receive this right over 10MHz bands for three years via regional auctions; and finally the rest of the population, who can access and utilize a portion of this band as long as they first check with a central database of registered users. CBRS, along with standardization efforts to extend mobile cellular networks to operate in the unlicensed bands, opens the door for private cellular networks similar to Wi-Fi. This is proving especially attractive to enterprises looking to establish a *Private 5G* service.

The specific frequency bands that are licensed for cellular networks vary around the world, and are complicated by the fact that network operators often simultaneously support both old/legacy technologies and new/next-generation technologies, each of which occupies a different frequency band. The high-level summary is that traditional cellular technologies range from 700-2400 MHz, with new mid-spectrum allocations now happening at 6 GHz, and millimeter-wave (mmWave) allocations opening above 24 GHz.

While the specific frequency band is not directly relevant to understanding 5G from an architectural perspective, it does impact the physical-layer components, which in turn has indirect ramifications on the overall 5G system. We identify and explain these ramifications in later chapters, keeping in mind that ensuring the allocated spectrum is used *efficiently* is a critical design goal.

Finally, in addition to the long-established 3GPP standards body and the set of national regulatory agencies around the world, a new organization—called the *Open-RAN Alliance (O-RAN)* —has recently been established to focus on “opening up the Radio Access Network”. We’ll see specifically what this means and how the O-RAN differs from the 3GPP in Chapter 4, but for now, its existence highlights an important dynamic in the industry: 3GPP has become a vendor-dominated organization, with network operators (AT&T and China Mobile were the founding members) creating O-RAN to break vendor lock-in.

1.2 1.2 Access Networks

The mobile cellular network is part of the access network that implements the Internet’s so-called *last mile*. (Another common access technology is *Passive Optical Networks (PON)*, colloquially known as Fiber-to-the-Home.) These mobile access networks have historically been provided by both big and small *Mobile Network Operators (MNOs)*. Global MNOs such as AT&T run access networks at thousands of aggregation points of presence across a country such as the US, along with a national backbone that interconnects those sites. Small regional and municipal MNOs might run an access network with one or two points of presence, and then connect to the rest of the Internet through some large operator’s backbone.

As illustrated in Figure 1.1, access networks are physically anchored at thousands of aggregation points of presence within close proximity to end users, each of which serves anywhere from 1,000-100,000 subscribers, depending on population density. In practice, the physical deployment of these “edge” locations vary from operator to operator, but one possible scenario is to anchor both the cellular and wireline access networks in Telco *Central Offices*.

Historically, the Central Office—officially known as the *PSTN (Public Switched Telephone Network) Central Office*—anchored wired access (both telephony and broadband), while the cellular network evolved independently by deploying a parallel set of *Mobile Telephone Switching Offices (MTSO)*. Each MTSO

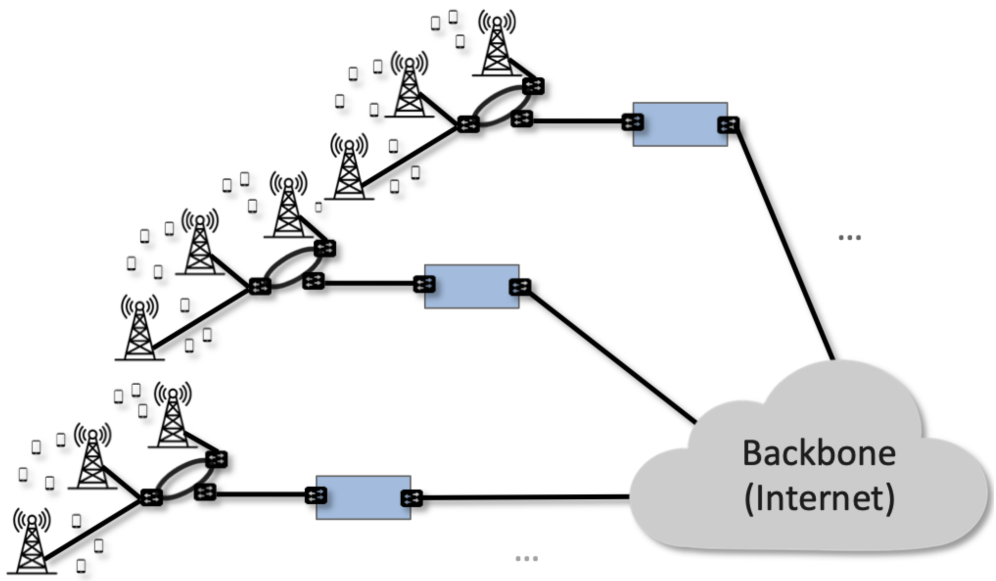

Figure 1.1.: A global mobile network built by first aggregating traffic from hundreds of wireless base stations, and then interconnecting those aggregation points over the Internet.

serves as a *mobile aggregation* point for the set of cell towers in a given geographic area. For our purposes, the important idea is that such aggregation points exist, and it is reasonable to think of them as defining the edge of an operator-managed access network. For simplicity, we sometimes use the term "Central Office" as a synonym for both types of edge sites.

Finally, one aspect of the mobile network that may not be obvious from Figure 1.1 is that it supports global connectivity, independent of the Internet (which is technically just one of many available backbone technologies). That is, the cellular network supports a universal addressing scheme, similar in principle (but significantly different in details) from the Internet's universal IP-based addressing scheme. This addressing scheme makes it possible to establish a voice call between any two cell phones, but of course, IP addresses still come into play when trying to establish a data (broadband) connection to/from a cell phone or other mobile device. Understanding the relationship between mobile addresses and IP addresses is a topic we will explore in later chapters.

1.3 1.3 Managed Cloud Service

The previous section gives a decidedly Telco-centric view of the mobile cellular network, which makes sense because Telcos have been the dominant MNOs for the past 40+ years. But with 5G's focus on broadening the set of services it supports, and embracing general platforms that can host yet-to-be-invented applications, the mobile cellular network is starting to blur the line between the access network and the cloud.

5G, Wi-Fi, and the Role of Spectrum

WiFi networks use unlicensed radio spectrum that do not require WiFi network operators to get advance regulatory approval. At the same time, anyone can access the same spectrum, subject to limits on transmission power. As a result, WiFi networks share their bands with devices including baby monitors, cordless phones, etc., so the WiFi MAC layer assumes the presence of physical-layer interference. Enterprise WiFi deployments, such as those on college campuses and in corporate office buildings, *perform more centralized management of interference across multiple overlapping access points, but risk of interference remains and thus the service remains best-effort.*

Cellular access networks typically use licensed spectrum that is owned or leased by the carrier for long periods of time at high cost. Even "lightly licensed" spectrum such as CBRS offers more control over interference than Wi-Fi. Since the cellular radio has exclusive access to spectrum over a geographic region, cellular waveforms are designed for wide-area coverage and high spectral efficiency. Managing access to the spectrum, as we shall see, is an important aspect of the 5G architecture.

Many of the differences between 5G and Wi-Fi follow from the differences in spectrum and radio characteristics. For example, cellular deployments, with the expense of spectrum being a given, have historically been carried out by well-resourced actors who can acquire land, build and connect towers, and hire skilled staff. However, the rise of enterprise 5G and the availability of lightly licensed spectrum is leading to a blurring of the lines between the two approaches.

The rest of this book explains what that means in detail. As an overview, thinking of 5G connectivity as a cloud service means that instead of using purpose-built devices and telephony-based operational practices to deliver mobile connectivity, the 5G network is built from commodity hardware, software-defined networks, and cloud-based operational practices. And, just as with familiar cloud applications, the end result is a system that increases both feature velocity and operational uniformity. These advantages are available to legacy MNOs, but whether they will fully embrace them is yet to be seen, so we do not limit ourselves to existing stakeholders or business models. In particular, this book focuses on how enterprises can be their own MNOs, or alternatively, acquire private 5G connectivity as a managed cloud service from non-traditional MNOs.

To this end, Figure 1.2 depicts a simplified Private 5G deployment that the rest of this book works toward. At a high level, the figure shows a wide range of enterprise use cases that might take advantage of 5G connectivity, with the data plane of the 5G service running on-prem (on an edge cloud running within the enterprise), and the control plane of the 5G service running off-prem (in the global cloud).[1] Enterprise administrators control their service through a management console, much in the same way they might log into an AWS, GCP, or Azure console to control a cloud-based storage or compute service. Finally, applications are distributed across both edge and centralized clouds, taking advantage of what is commonly referred to as a *hybrid cloud*.

Hosting a 5G connectivity service on an edge cloud is perfectly aligned with one of the most pronounced trends in cloud computing: moving elements of the cloud from the datacenter to locations that are in close proximity to end users and their devices. Before looking at how to realize 5G on an edge cloud, we start by considering why edge clouds are gaining momentum in the first place.

The cloud began as a collection of warehouse-sized datacenters, each of which provided a cost-effective way to power, cool, and operate a scalable number of servers. Over time, this shared infrastructure lowered the barrier to deploying scalable Internet services, but today, there is increasing pressure to offer low-latency/high-bandwidth cloud applications that cannot be effectively implemented in remote datacenters. Augmented Reality (AR), Virtual Reality (VR), Internet of Things (IoT), and Autonomous Vehicles are all examples of this kind of application. Such applications benefit from moving at least part of their functionality out of the datacenter and towards the edge of the network, closer to end users.

The idea of such deployments is to first collect operational data on assets and infrastructure, from sensors, video feeds and telemetry from machinery. It then applies Machine Learning (ML) or other forms of analysis to this data to gain insights, identify patterns and predict outcomes (e.g., when a device is likely to fail). The final step is to automate industrial processes so as to minimize human intervention and

[1] We use the term "data plane" in the generic sense in this description. As we'll see in Chapter 2, the 5G architecture refers to it as "user plane".

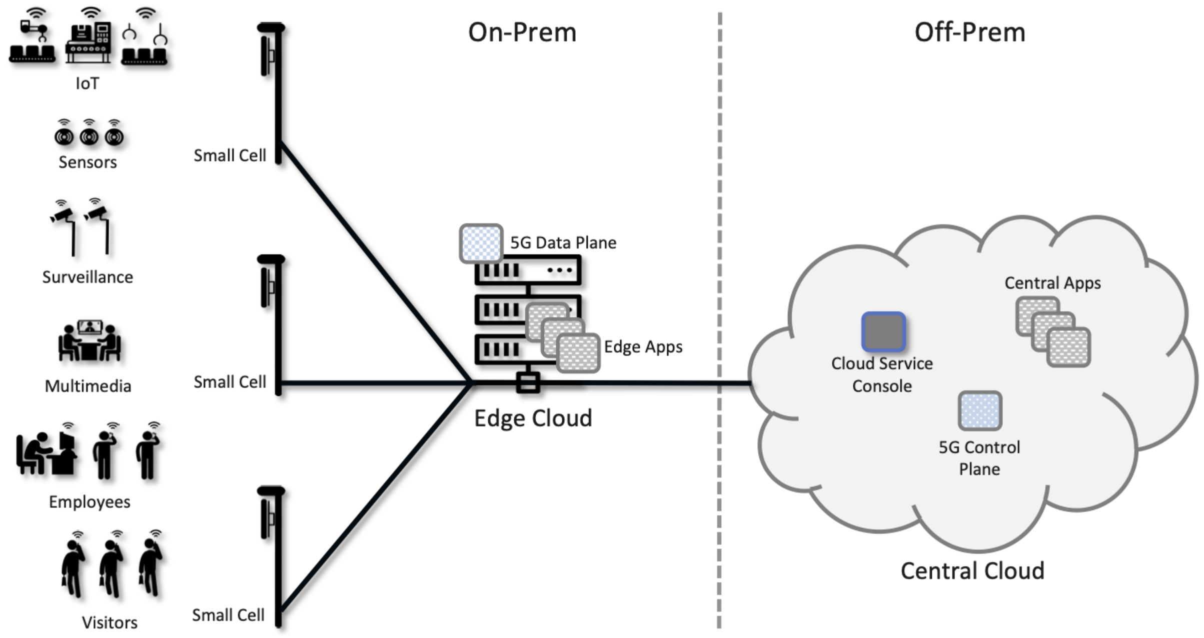

Figure 1.2.: Enterprise-based deployment of 5G connectivity, running as a managed cloud service.

enable remote operations (e.g., power optimization, idling quiescent machinery). The overall goal is to create an IT foundation for continually improving industrial operations through software.

But precisely where this edge is *physically* located depends on who you ask. If you ask a network operator that already owns and operates thousands of Central Offices, then their Central Offices are an obvious answer. Others might claim the edge is located at the 14,000 Starbucks locations (for example) across the US, and still others might point to the tens of thousands of cell towers spread across the globe. Our approach is to be location agnostic, but to make the discussion concrete, we use enterprises as our exemplar deployment.

At the same time cloud providers started pursuing edge deployments, network operators began to re-architect their access network to use the same commodity hardware and best practices in building scalable software as the cloud providers. Such a design, which is sometimes referred to as CORD *(Central Office Re-architected as a Datacenter)*, supports both the access network and edge services co-located on a shared cloud platform. This platform is then replicated across hundreds or thousands of operator sites, including Central Offices.

Traditional network operators did this because they wanted to take advantage of the same economies of scale and feature velocity as cloud providers. CORD gave them a general architecture to work towards, but also an open source Kubernetes-based reference implementation to model their solutions on. That original implementation of CORD is the direct predecessor to the Aether platform that we use as a reference implementation in this book.

Further Reading

L. Peterson, *et al*. Central Office Re-architected as a Datacenter.. IEEE Communications, October 2016.

A.D. Little Report. Who Dares Wins! How Access Transformation Can Fast-Track Evolution of Operator Production Platforms. September 2019.

An important takeaway from this discussion is that to understand how 5G is being implemented, it is helpful to have a working understanding of how clouds are built. This includes the use of *commodity* *hardware* (both servers and bare-metal switches), horizontally scalable *microservices* (also referred to as *cloud native*), and *Software-Defined Networks (SDN)*. It is also helpful to have an appreciation for how cloud software is developed, tested, deployed, and operated, including practices such as *DevOps* and *Continuous Integration / Continuous Deployment (CI/CD)*. We recommend two companion books to help fill the gaps in your understanding of these foundational technologies.

Further Reading

Software-Defined Networks: A Systems Approach. November 2021.

Edge Cloud Operations: A Systems Approach. June 2022.

1.4 1.4 Beyond 5G

From the moment MNOs started rolling out 5G in 2019, people started talking about what comes next. The obvious answer is 6G, but it's not at all clear that the decadal generations of the past 40 years will continue into the future. Today, you hear alternatives like "NextG" and "Beyond 5G" more often than 6G, which could be a sign that the industry is undergoing a fundamental shift. And there is an argument that we're in the midst of a sea change that will render the generational distinction largely meaningless. There are two complementary reasons for this, both at the heart of what's important about Private 5G.

The first factor is that by adopting cloud technologies, the mobile cellular network is hoping to cash in on the promise of feature velocity. This "agility" story was always included in the early 5G promotional material, as part of the case for why a 5G upgrade would be a worthwhile investment, but the consequence of those technologies now finding their way into the mainstream is that new features can be introduced rapidly and deployed continuously. At some point, the frequency of continual improvements renders generational distinctions irrelevant.

The second factor is that agility isn't only about cadence; it's also about customization. That is, these changes can be introduced bottom-up—for example by enterprises and their edge cloud partners in the case of Private 5G—without necessarily depending on (or waiting for) a global standardization effort. If an enterprise finds a new use case that requires a specialized deployment, only its Private 5G deployment needs to adopt the necessary changes. Reaching agreement with all the incumbent stakeholders will no longer be a requirement.

It's anyone's guess where this will take us, but it will be interesting to see how this dynamic impacts the role of standardization: what aspects of the mobile network require global agreement and what aspects do not because they can evolve on a case-by-case basis. While standards often spur innovation (TCP and HTTP are two great examples from the Internet experience), sometimes standards actually serve as a barrier to competition, and hence, innovation. Now that software is eating the mobile cellular network—with Private 5G deployed in enterprises likely setting the pace—we will learn which standards are which.

In summary, that 5G is on an evolutionary path is the central theme of this book. We call attention to its importance here, and revisit the topic throughout the book. We are writing this book for *system generalists*, with the goal of helping bring a community that understands a broad range of systems issues (but knows little or nothing about the cellular network) up to speed so they can play a role in its evolution. This is a community that understands both feature velocity and best practices in building robust scalable systems, and so has an important role to play in bringing all of 5G's potential to fruition.

CHAPTER

TWO

CHAPTER 2: ARCHITECTURE

This chapter identifies the main architectural components of the mobile cellular network. We need to introduce some terminology to do this, which can be confusing for those whose networking background comes from the Internet. This is partly because some of what needs to happen in a mobile network, such as keeping track of which base station is serving a given mobile device, doesn't really have a parallel in fixed networks. On top of that, the terminology came out of the 3GPP standardization process, which was historically concerned with telephony and almost completely disconnected from the IETF and other Internet-related efforts. To further confuse matters, 3GPP terminology often changes with each generation (e.g., a base station is called eNB in 4G and gNB in 5G). We address situations like this by using generic terminology (e.g., base station), and referencing the 3GPP-specific counterpart only when the distinction is helpful. This example is only the tip of the terminology iceberg. Marcin Dryjanski's blog post gives a broader perspective on the complexity of terminology in 5G.

Further Reading

Marcin Dryjanski. LTE and 5G Differences: System Complexity. July 2018.

2.1 2.1 Overview

The mobile cellular network provides wireless connectivity to devices that are (potentially) on the move. These devices, which are known as *User Equipment (UE)*, have traditionally corresponded to mobile phones and tablets, but increasingly include cars, drones, industrial and agricultural machines, robots, home appliances, medical devices, and so on. In some cases, the UEs may be devices that do not move, e.g., router interfaces using cellular connectivity to provide broadband access to remote dwellings.

As shown in Figure 2.1, the mobile cellular network consists of two main subsystems: the *Radio Access Network (RAN)* and the *Mobile Core*. The RAN manages the radio resources (i.e., spectrum), making sure it is used efficiently and meets the quality of service (QoS) requirements of every user. It corresponds to a distributed collection of base stations. As noted above, these are cryptically named *eNodeB* or *eNB* (which is short for *evolved Node B*) in 4G. In 5G, base stations are known as *gNB*, where the "g" stands for *next Generation*.

The Mobile Core is a bundle of functionality (conventionally packaged as one or more devices) that serves several purposes.

- Authenticates devices prior to attaching them to the network
- Provides Internet (IP) connectivity for both data and voice services.
- Ensures this connectivity fulfills the promised QoS requirements.

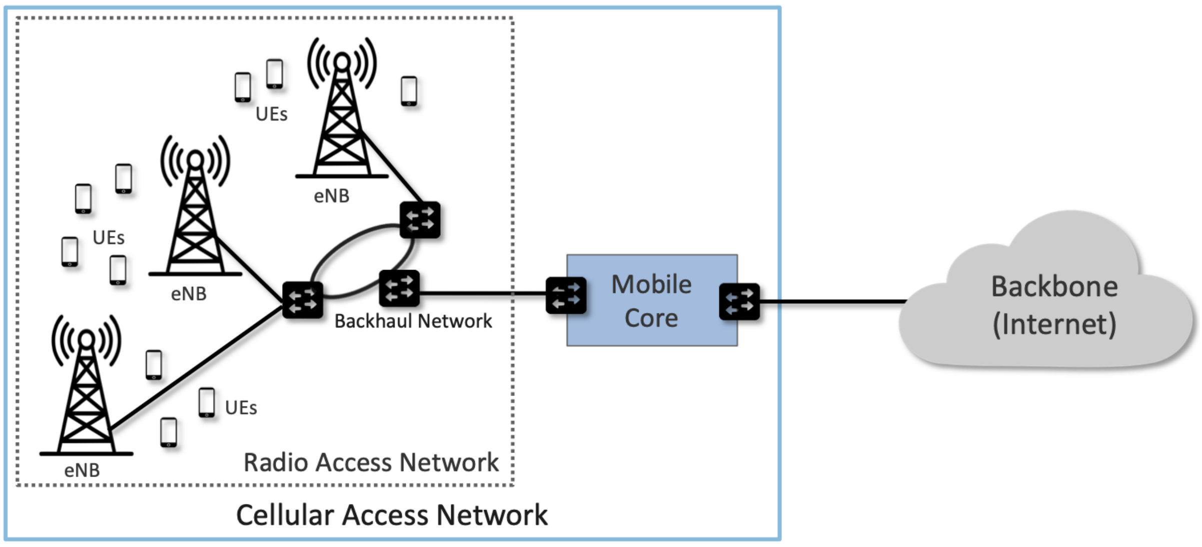

Figure 2.1.: Mobile cellular networks consist of a Radio Access Network (RAN) and a Mobile Core.

- Tracks user mobility to ensure uninterrupted service.
- Tracks subscriber usage for billing and charging.

For readers familiar with the Internet architecture and Wi-Fi as a common access technology, some of these functions might look a bit surprising. For example, Wi-Fi, like most of the Internet, normally provides a best-effort service, whereas cellular networks often aim to deliver some sort of QoS guarantee. Tracking subscribers for both mobility and billing are also not the sort of things we tend to think about in the Internet, but they are considered important functions for cellular networks. The reasons for these differences are numerous, including the typically large costs of acquiring cellular spectrum and maintaining the infrastructure to use it such as radio towers. With that large investment, there is a desire to recoup costs by charging subscribers, which in turn leads to making some sort of service guarantees to those subscribers to justify the cost. There is also a need to maximize the efficiency of spectrum usage. Much of the complexity of the mobile core follows from these requirements being imposed by service providers. Even when we get to enterprises running their own 5G networks, they still need to manage the usage of spectrum to obtain the benefits of 5G over Wi-Fi, such as more predictable control over latency and bandwidth.

Note that Mobile Core is another example of a generic term. In 4G it was called the *Evolved Packet Core (EPC)* and in 5G it is called the *5G Core (5GC)*. Moreover, even though the word “Core” is in its name, the Mobile Core runs near the edge of the network, effectively providing a bridge between the RAN in some geographic area and the greater IP-based Internet. 3GPP provides significant flexibility in how the Mobile Core is geographically deployed, ranging from minimal deployments (the RAN and the mobile core can be co-located) to areas that are hundreds of kilometers wide. A common model is that an instantiation of the Mobile Core serves a metropolitan area. The corresponding RAN would then span several dozens (or even hundreds) of cell towers in that geographic area.

Taking a closer look at Figure 2.1, we see that a *Backhaul Network* interconnects the base stations that implement the RAN with the Mobile Core. This network is typically wired, may or may not have the ring topology shown in the figure, and is often constructed from commodity components found elsewhere in the Internet. For example, the *Passive Optical Network (PON)* that implements Fiber-to-the-Home is a prime candidate for implementing the RAN backhaul, with the RAN effectively running as an *overlay* on top of whatever technology is used. Switched ethernet, such as you might find in an enterprise, is another suitable choice. The backhaul network is obviously a necessary part of the RAN, but it is an implementation choice and not prescribed by the 3GPP standard.

Although 3GPP specifies all the elements that implement the RAN and Mobile Core in an open standard—including sub-layers we have not yet introduced—network operators have historically bought proprietary implementations of each subsystem from a single vendor. This lack of an open source implementation contributes to the perceived "opaqueness" of the mobile cellular network in general, and the RAN in particular. And while it is true that base stations contain sophisticated algorithms for scheduling transmission on the radio spectrum—algorithms that are considered valuable intellectual property of the equipment vendors—there is significant opportunity to open and disaggregate both the RAN and the Mobile Core. This book gives a recipe for how to do exactly that.

Before getting to those details, we have three more architectural concepts to introduce. First, Figure 2.2 redraws components from Figure 2.1 to highlight the fact that a base station has an analog component (depicted by an antenna) and a digital component (depicted by a processor pair). This book mostly focuses on the latter, but we introduce enough information about the over-the-air radio transmission to appreciate its impact on the overall architecture.

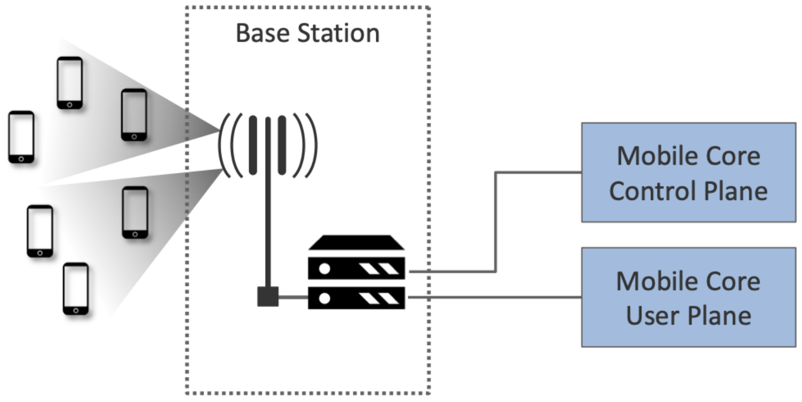

Figure 2.2.: Mobile Core divided into a Control Plane and a User Plane, an architectural feature known as CUPS: Control and User Plane Separation.

The second concept, also depicted in Figure 2.2, is to partition the Mobile Core into a *Control Plane* and *User Plane*. This is similar to the control/data plane split that anyone familiar with the Internet would recognize, and draws in particular on the ideas of software-defined networking (SDN) by placing control and user planes in separate devices. 3GPP has introduced a corresponding acronym—*CUPS, Control and User Plane Separation*—to denote this idea. One motivation for CUPS is to enable control plane resources and data plane resources to be scaled independently of each other.

Finally, one of the key aspirational goals of 5G is the ability to segregate traffic for different usage domains into isolated *network slices*, each of which delivers a different level of service to a collection of devices and applications. Thinking of a network slice as a wireless version of a virtual network is a fair approximation, although as we'll see in later chapters, the implementation details differ.

For example, Figure 2.3 shows two slices, one supporting IoT workloads and the other supporting multimedia streaming traffic. As we'll see throughout the book, an important question is how slicing is realized end-to-end, across the radio, the RAN, and the Mobile Core. This is done through a combination of allocating distinct resources to each slice and scheduling shared resources on behalf of a set of slices.

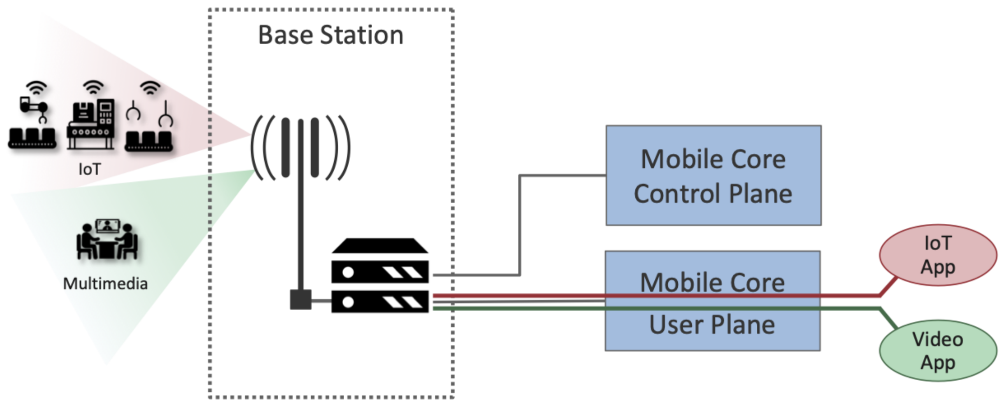

Figure 2.3.: Different usage domains (e.g., IoT and Video Streaming) instantiate distinct *network slices* to connect a set of devices with one or more applications.

2.2 2.2 Radio Transmission

Before describing the RAN and Mobile Core subsystems, we first call attention to the obvious: that the base stations that comprise the RAN communicate with UEs via electromagnetic radio waves. This book is not about the physics of this over-the-air communication, and only skims the surface of the information theory that underlies it. But identifying the abstract properties of wireless communication is an essential foundation for understanding the rest of the 5G architecture.

If you imagine the base stations as implementing a multi-layer protocol stack (which, as we'll see in Chapter 4, they do), then radio transmission is the responsibility of the bottom-most layers of that stack, where (a) digital/analog conversion happens, and (b) analog radio waves are transmitted/received. Chapter 3 introduces radio transmission with enough specificity to lay the necessary foundation, so we're able to understand all the layers that come above it.

For the purposes of this chapter, we only need to understand the following. First, the RAN is responsible for managing how the radio spectrum is shared among thousands of UEs connected to hundreds of base stations in a geographic region. The primary purpose of Chapter 3 is to establish an abstract interface by which the RAN can manage that spectrum without having to worry about the details of waveforms, modulation, or coding algorithms. All important topics, to be sure, but in the realm of information theory rather than system design that is the focus of this book.

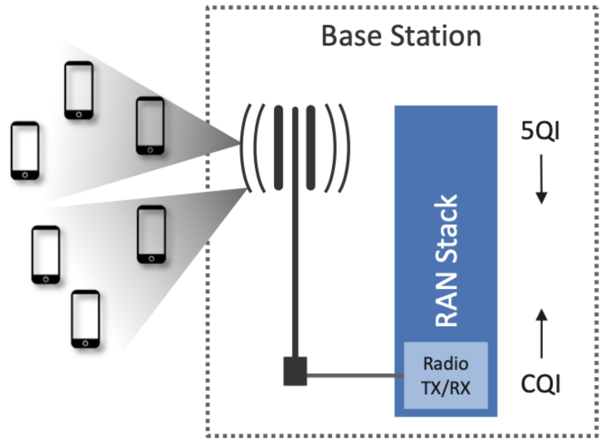

Figure 2.4.: Abstractly, measures of signal quality (CQI) and declarations of intended data delivery quality (5QI) are passed up and down the RAN stack.

Second, there are two important pieces of information shared between the higher layers of the base station protocol stack that manages the RAN as a whole, and the lower layers of the stack that manage radio transmissions on a particular base station. One is the signal-to-noise ratio that the base station observes when communicating with each UE. This is called the *Channel Quality Indicator (CQI)* and it is passed *up* from the radio. The other is the quality of service the network wants to give a particular UE. This is called the *5G QoS Identifier (5QI)* and it is passed *down* to the radio. This abstract summary, as shown in Figure 2.4, is sufficient to introduce the RAN and Mobile Core. We will fill in more details about both of these parameters in Chapter 3.

Uniqueness of Wireless Links

While it is common in networking to abstract the link layer by treating the link as something that just delivers packets at some rate from point A to point B, there are important differences between wireless links and wired links that cannot be entirely abstracted away at higher layers. This is especially true when mobile devices are involved, as the quality of a link will vary depending on the distance between transmitter and receiver, the relative velocity of the endpoints, reflections of radio waves from other objects, and interference from other transmitters. All of these factors come into play in determining the Channel Quality Indicator (CQI).

Further complicating the picture in a mobile network is that a given UE is often within reach of more than one base station, presenting the option to handover the UE from one base station to another. The decision to do so is not just a matter of picking the base station with the best channel quality, but rather a matter of trying to optimize the whole system, in which the goal is to support as many UEs as possible at the desired quality level given the available spectrum and coverage.

Finally, like the rest of the mobile cellular network, the radio comes with a set of acronyms, with *LTE (Long-Term Evolution)* and *NR (New Radio)* being the two most widely known. These are marketing terms commonly associated with the radio technology for 4G and 5G, respectively. They are important only in the sense that many of the new features promised by 5G can be directly attributed to improvements in the underlying radio technology. For our purposes, the key is the set of new *use cases* the upgraded radio technology enables, and why. We introduce these improvements to the radio in Chapter 3, and tie them to the use cases they enable. Subsequent chapters will then explain how the RAN and Mobile Core need to evolve so as to deliver on this potential.

2.3 2.3 Radio Access Network

We now describe the RAN by sketching the role each base station plays. Keep in mind this is like describing the Internet by explaining how a router works—not an unreasonable place to start, but it doesn't fully do justice to the end-to-end story.

First, each base station establishes the wireless channel for a subscriber's UE upon power-up or upon handover when the UE is active. This channel is released when the UE remains idle for a predetermined period of time. Using 3GPP terminology, this wireless channel is said to provide a *radio bearer*. The term "bearer" has historically been used in telecommunications (including early wireline technologies such as ISDN) to denote a data channel, as opposed to a channel that carries signaling information.

Second, each base station establishes "3GPP Control Plane" connectivity between the UE and the corresponding Mobile Core Control Plane component, and forwards signaling traffic between the two. This signaling traffic enables UE authentication, registration, and mobility tracking.

Third, for each active UE, the base station establishes one or more tunnels to the corresponding Mobile

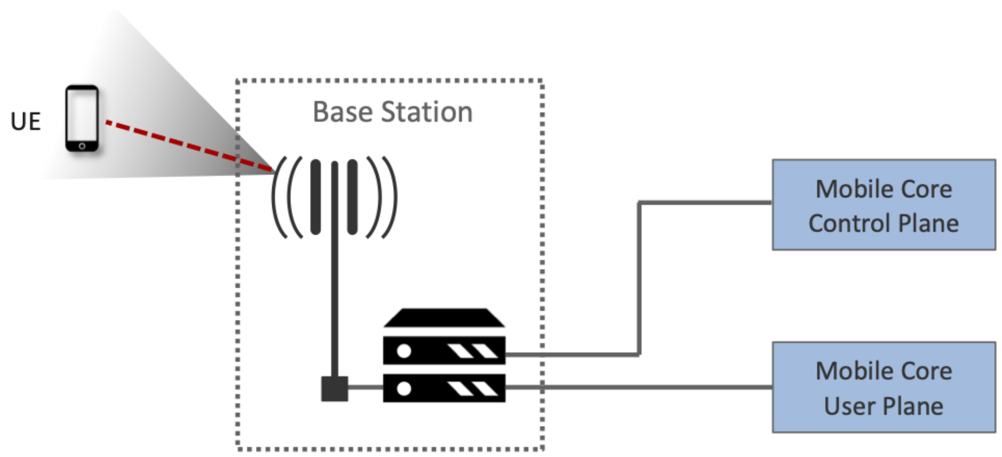

Figure 2.5.: UE detects (and connects to) base station.

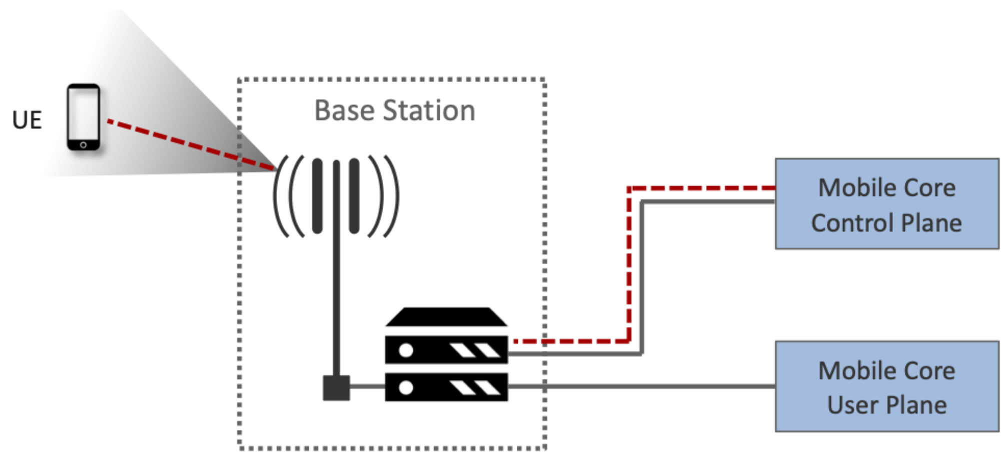

Figure 2.6.: Base Station establishes control plane connectivity between each UE and the Mobile Core.

Core User Plane component. Figure 2.7 shows just two (one for voice and one for data), and while in practice 4G was limited to just two, 5G aspires to support many such tunnels as part of a generalized network slicing mechanism.

Fourth, the base station forwards both control and user plane packets between the Mobile Core and the UE. These packets are tunneled over SCTP/IP and GTP/UDP/IP, respectively. SCTP (Stream Control Transport Protocol) is an alternative reliable transport to TCP, tailored to carry signaling (control) information for telephony services. GTP (a nested acronym corresponding to (General Packet Radio Service) Tunneling Protocol) is a 3GPP-specific tunneling protocol designed to run over UDP.

It is noteworthy that connectivity between the RAN and the Mobile Core is IP-based. This was introduced as one of the main changes between 3G and 4G. Prior to 4G, the internals of the cellular network were circuit-based, which is not surprising given its origins as a voice network. This also helps to explain why in Section 2.1 we characterized the RAN Backhaul as an overlay running on top of some Layer 2 technology.

Fifth, each base station coordinates UE handovers with neighboring base stations, using direct station-to-station links. Exactly like the station-to-core connectivity shown in the previous figure, these links are used to transfer both control plane (SCTP over IP) and user plane (GTP over UDP/IP) packets. The decision as to when to do a handover is based on the CQI values being reported by the radio on each of the base stations within range of the UE, coupled with the 5QI value those base stations know the RAN has promised to deliver to the UE.

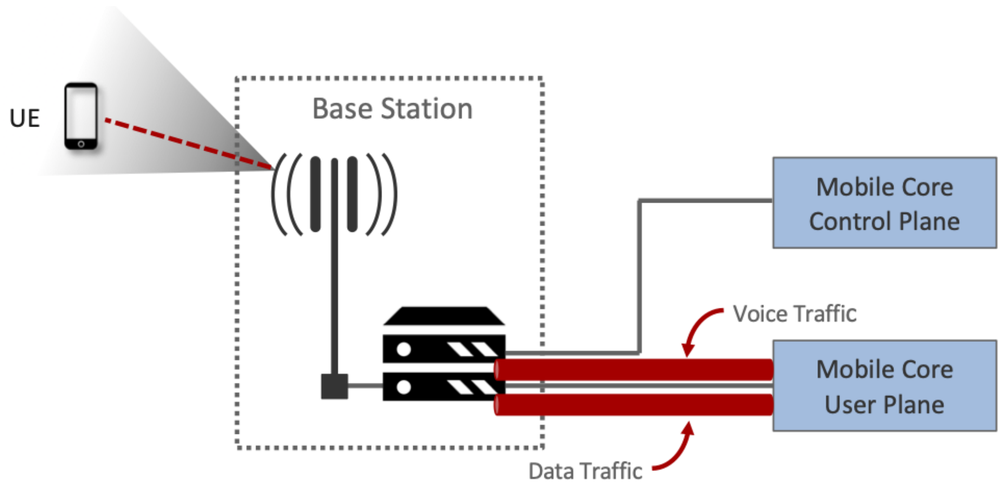

Figure 2.7.: Base station establishes one or more tunnels between each UE and the Mobile Core's User Plane (known in 3GPP terms as PDU session).

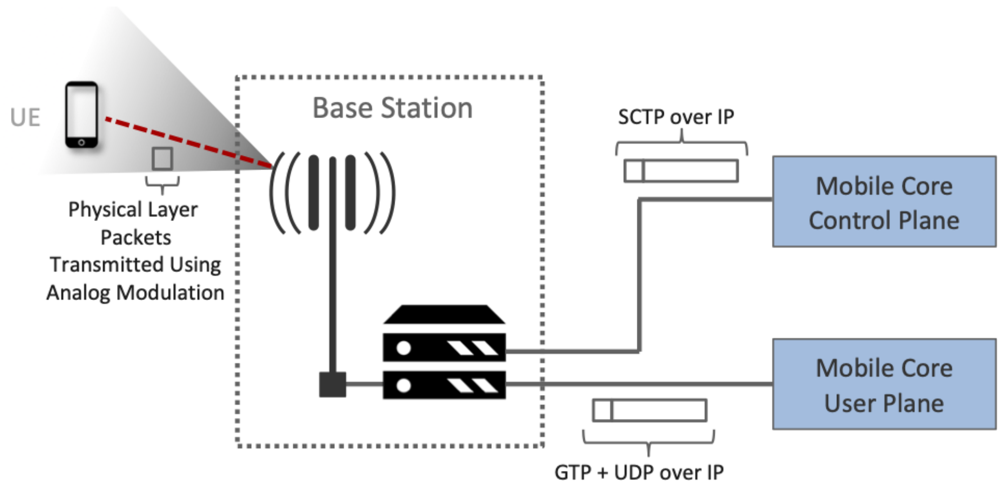
UE
Base Station
SCTP over IP
Mobile Core Control Plane
Physical Layer Packets Transmitted Using Analog Modulation
Mobile Core User Plane
GTP + UDP over IP

Figure 2.8.: Base Station to Mobile Core (and Base Station to Base Station) control plane tunneled over SCTP/IP and user plane tunneled over GTP/UDP/IP.

Sixth, the base stations coordinate wireless multi-point transmission to a UE from multiple base stations, which may or may not be part of a UE handover from one base station to another.

The main takeaway is that the base station can be viewed as a specialized forwarder. In the Internet-to-UE direction, it fragments outgoing IP packets into physical layer segments and schedules them for transmission over the available radio spectrum, and in the UE-to-Internet direction it assembles physical layer segments into IP packets and forwards them (over a GTP/UDP/IP tunnel) to the upstream user plane of the Mobile Core. Also, based on observations of the wireless channel quality and per-subscriber policies, it decides whether to (a) forward outgoing packets directly to the UE, (b) indirectly forward packets to the UE via a neighboring base station, or (c) utilize multiple paths to reach the UE. The third case has the option of either spreading the physical payloads across multiple base stations or across multiple carrier frequencies of a single base station (including Wi-Fi).

In other words, the RAN as a whole (i.e., not just a single base station) not only supports handovers (an obvious requirement for mobility), but also *link aggregation* and *load balancing*, mechanisms that are similar to those found in other types of networks. These functions imply a global decision-making process, whereby it's possible to forward traffic to a different base station (or to multiple base stations)

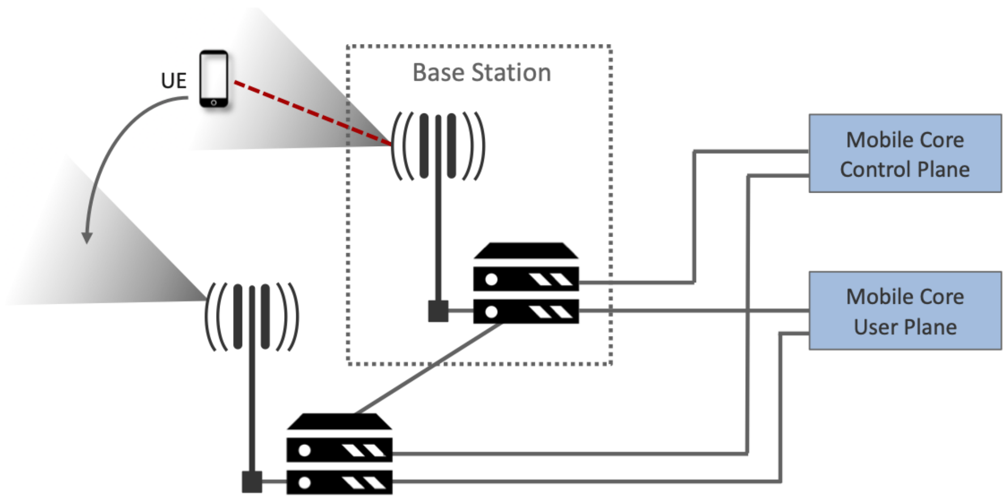

Figure 2.9.: Base Stations cooperate to implement UE hand over.

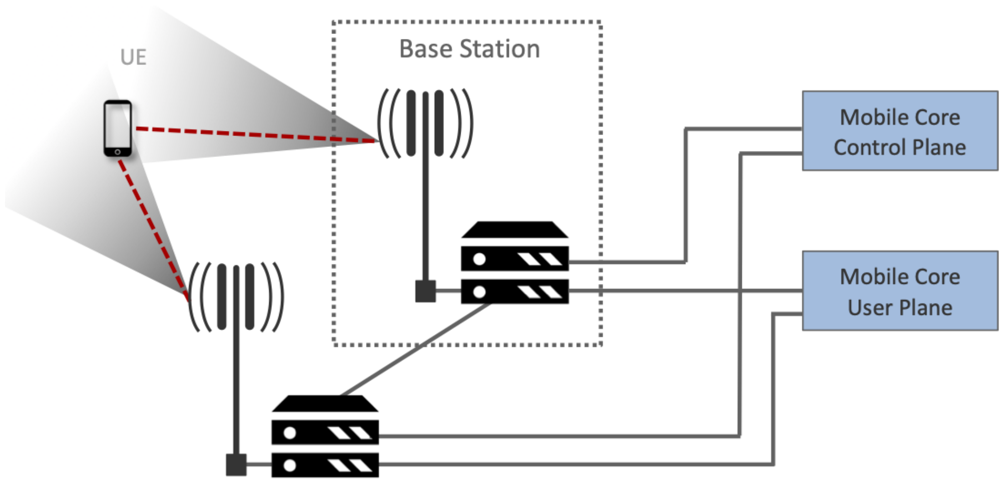

Figure 2.10.: Base Stations cooperate to implement multipath transmission (link aggregation) to UEs.

in an effort to make efficient use of the radio spectrum over a larger geographic area. We will revisit how such RAN-wide (global) decisions can be made using SDN techniques in Chapter 4.

2.4 2.4 Mobile Core

At the most basic level, the function of the Mobile Core is to provide packet data network connectivity to mobile subscribers, i.e., connect them to the Internet. (The mobile network assumes that multiple packet data networks can exist, but in practice the Internet is the one that matters). As we noted above, there is more to providing this connectivity than meets the eye: the Mobile Core ensures that subscribers are authenticated and aims to deliver the service qualities to which they have subscribed. As subscribers may move around among base station coverage areas, the Mobile Core needs to keep track of their whereabouts at the granularity of the serving base station. The reasons for this tracking are discussed further in Chapter 5. It is this support for security, mobility, and QoS that differentiates the cellular network from Wi-Fi.

We start with the security architecture, which is grounded in two trust assumptions. First, each base station trusts that it is connected to the Mobile Core by a secure private network, over which it establishes the tunnels introduced in Figure 2.8: a GTP/UDP/IP tunnel to the Core's User Plane (Core-UP) and a

SCTP/IP tunnel to the Core's Control Plane (Core-CP). Second, each UE has an operator-provided SIM card, which contains information that uniquely identifies the subscriber and includes a secret key that the UE uses to authenticate itself.

The identifier burned into each SIM card, called an *IMSI (International Mobile Subscriber Identity)*, is a globally unique identifier for every device connected to the global mobile network. Each IMSI is a 64-bit, self-describing identifier, which is to say, it includes a *Format* field that effectively serves as a mask for extracting other relevant fields. For example, the following is the interpretation we assume in this book (where IMSIs are commonly represented as an up to 15-digit decimal number):

- **MCC:** Mobile Country Code (3-digit decimal number).
- **MNC:** Mobile Network Code (2 or 3-digit decimal number).
- **ENT:** Enterprise Code (3-digit decimal number).
- **SUB:** Subscriber (6-digit decimal number).

The first two fields (*MCC*, *MNC*) are universally understood to uniquely identify the MNO, while that last two fields are one example of how an MNO might use additional hierarchical structure to uniquely identify every device it serves. We are working towards delivering 5G connectivity to enterprises (hence the *ENT* field), but other MNOs might assign the last 9 digits using some other structure.

The *MCC/MNC* pair—which is also called the *Public Land Mobile Network (PLMN)* identifier—plays a role in roaming: when a UE tries to connect to a "foreign network" those fields are used to find the "home network", where the rest of the IMSI leads to a subscriber profile that says whether or not roaming is enabled for this device. The following walks through what happens when a device connects to its home network; more information about the global ramifications is given at the end of the section.

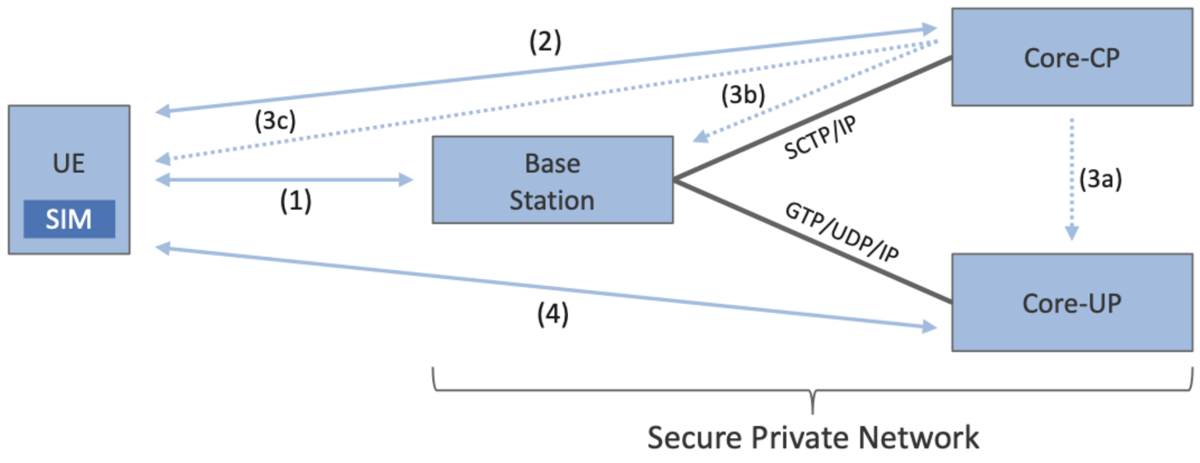

Figure 2.11.: Sequence of steps to establish secure Control and User Plane channels.

Figure 2.11 shows the per-UE connection sequence. When a UE first becomes active, it communicates with a nearby base station over a temporary (unauthenticated) radio link (Step 1). The base station forwards the request to the Core-CP over the existing SCTP connection, and the Core-CP (assuming it recognizes the IMSI) initiates an authentication protocol with the UE (Step 2). 3GPP identifies a set of options for authentication and encryption, where the actual protocols used are an implementation choice. For example, *Advanced Encryption Standard* (AES) is one of the options for encryption. Note that this authentication exchange is initially in the clear since the base station to UE link is not yet secure.

Once the UE and Core-CP are satisfied with each other's identity, the Core-CP informs the other 5GC components of the parameters they will need to service the UE (Step 3). This includes: (a) instructing the Core-UP to initialize the user plane (e.g., assign an IP address to the UE and set the appropriate 5QI);

(b) instructing the base station to establish an encrypted channel to the UE; and (c) giving the UE the symmetric key it will need to use the encrypted channel with the base station. The symmetric key is encrypted using the public key of the UE (so only the UE can decrypt it, using its secret key). Once complete, the UE can use the end-to-end user plane channel through the Core-UP (Step 4).

There are three additional details of note about this process. First, the secure control channel between the UE and the Core-CP set up during Step 2 remains available, and is used by the Core-CP to send additional control instructions to the UE during the course of the session. In other words, unlike the Internet, the network is able to "control" the communication settings in edge devices.

Second, the user plane channel established during Step 4 is referred to as the *Default Data Radio Bearer*, but additional channels can be established between the UE and Core-UP, each with a potentially different 5QI. This might be done on an application-by-application basis, for example, based on policies present in the Core-CP for packets that require special/different treatment.

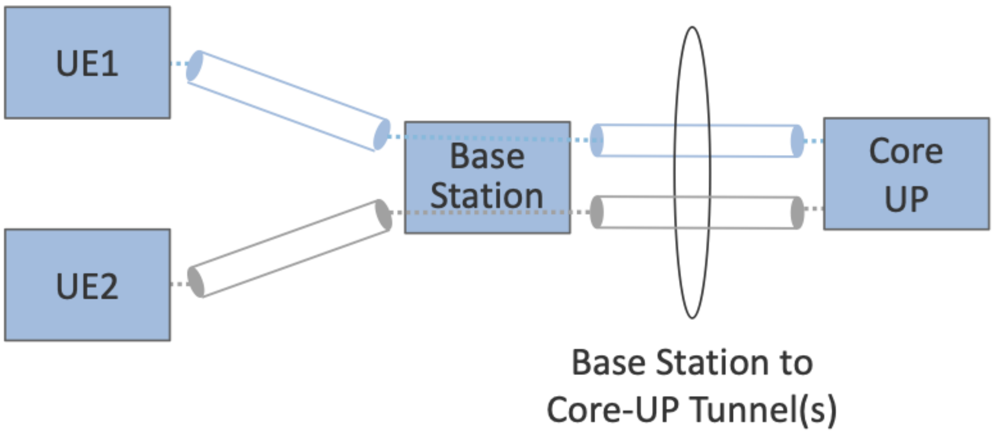

Figure 2.12.: Sequence of per-hop tunnels involved in an end-to-end User Plane channel.

In practice, these per-flow tunnels are often bundled into a single inter-component tunnel, which makes it impossible to differentiate the level of service given to any particular end-to-end UE channel. This is a limitation of 4G that 5G has ambitions to correct as part of its support for network slicing.

Support for mobility can now be understood as the process of re-executing one or more of the steps shown in Figure 2.11 as the UE moves throughout the RAN. The unauthenticated link indicated by (1) allows the UE to be known to all base stations within range. (We refer to these as *potential links* in later chapters.) Based on the signal's measured CQI, the base stations communicate directly with each other to make a handover decision. Once made, the decision is then communicated to the Mobile Core, re-triggering the setup functions indicated by (3), which in turn re-builds the user plane tunnel between the base station and the Core-UP shown in Figure 2.12. One of the most unique features of the cellular network is that the Mobile Core's user plane buffers data while idle UEs are transiting to active state, thereby avoiding dropped packets and subsequent end-to-end retransmissions.

In other words, the mobile cellular network maintains the *UE session* in the face of mobility (corresponding to the control and data channels depicted by (2) and (4) in Figure 2.11, respectively), but it is able to do so only when the same Mobile Core serves the UE (i.e., only the base station changes). This would typically be the case for a UE moving within a metropolitan area. Moving between metro areas—and hence, between Mobile Cores—is indistinguishable from power cycling a UE. The UE is assigned a new IP address and no attempt is made to buffer and subsequently deliver in-flight data. Independent of mobility, but relevant to this discussion, any UE that becomes inactive for a period of time also loses its session, with a new session established and a new IP address assigned when the UE becomes active again.

Note that this session-based approach can be traced to the mobile cellular network's roots as a connection-oriented network. An interesting thought experiment is whether the Mobile Core will continue to evolve so as to better match the connectionless assumptions of the Internet protocols that typically run on top of it.

We conclude this overview of the Mobile Core by returning to the role it plays in implementing a *global* mobile network. It is probably already clear that each MNO implements a database of subscriber information, allowing it to map an IMSI to a profile that records what services (roaming, data plane, hot spot support) the subscriber is paying for. This record also includes the international phone number for the device. How this database is realized is an implementation choice (of which we'll see an example in Chapter 5), but 3GPP defines an interface by which one Mobile Core (running on behalf of one MNO) queries another Mobile Core (running on behalf of some other MNO), to map between the IMSI, the phone number, and the subscriber profile.

2.5 2.5 Managed Cloud Service

The architectural overview presented up to this point focuses on the functional elements of the mobile cellular network. We now turn our attention to how this functionality is operationalized, and we do so in a decidedly software-defined and cloud-native way. This lays the foundation for the rest of the book, which builds towards supporting 5G connectivity as a managed cloud service. This is a marked change from the conventional Telco approach, whereby an operator bought purpose-built devices from a handful of vendors, and then managed them using the legacy OSS/BSS machinery that was originally designed for the telephony network.[1]

When we talk about "operationalizing" a network, we are referring to a substantial system that operators (whether they are traditional MNOs or cloud service providers) use to activate and manage all the constituent components (whether they are purpose-built devices or software running on commodity hardware). And because these network operators are people, one high-level summary is that this management layer (whether it is an OSS/BSS or a cloud orchestrator) provides a way to map high-level *Intents* onto low-level *Actions*.

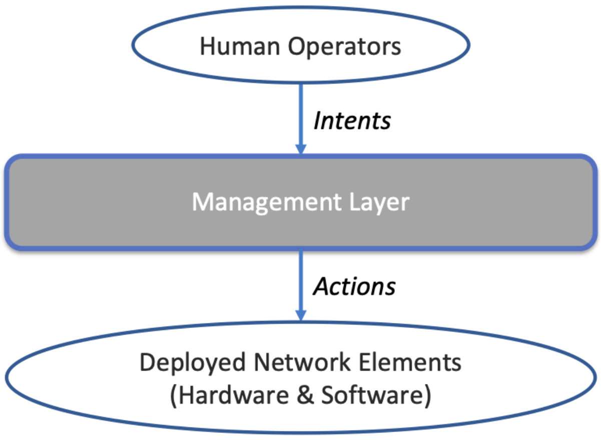

Figure 2.13.: High-level summary of the role operationalization plays in a network deployment.

This overview, as illustrated in Figure 2.13, is obviously quite abstract. To make the discussion more concrete, we use an open source implementation, called Aether, as an example. Aether is a Kubernetes-

[1] OSS/BSS stands for Operation Support System / Business Support System, and even traditional MNOs are now re-imagining them by adopting cloud practices. But this transition is a slow process due to all the legacy systems the Telcos need to continue supporting.

based edge cloud, augmented with a 5G-based connectivity service. Aether is targeted at enterprises that want to take advantage of 5G connectivity in support of edge applications that require predictable, low-latency connectivity. In short, "Kubernetes-based" means Aether is able to host container-based services, with Kubernetes being the platform used to orchestrate the services, and "5G-based connectivity" means Aether is able to connect those services to mobile devices throughout the enterprise's physical environment.

Aether supports this combination by implementing both the RAN and the user plane of the Mobile Core on-prem, as cloud-native workloads co-located on the Aether cluster. This is often referred to as *local breakout* because it enables direct communication between mobile devices and edge applications without data traffic leaving the enterprise, in contrast to what would happen with standard, operator-provided 5G service. This scenario is depicted in Figure 2.14, which shows unnamed edge applications running on-prem. Those edge applications might include the local processing of sensor data or control applications for the IoT devices, for example.

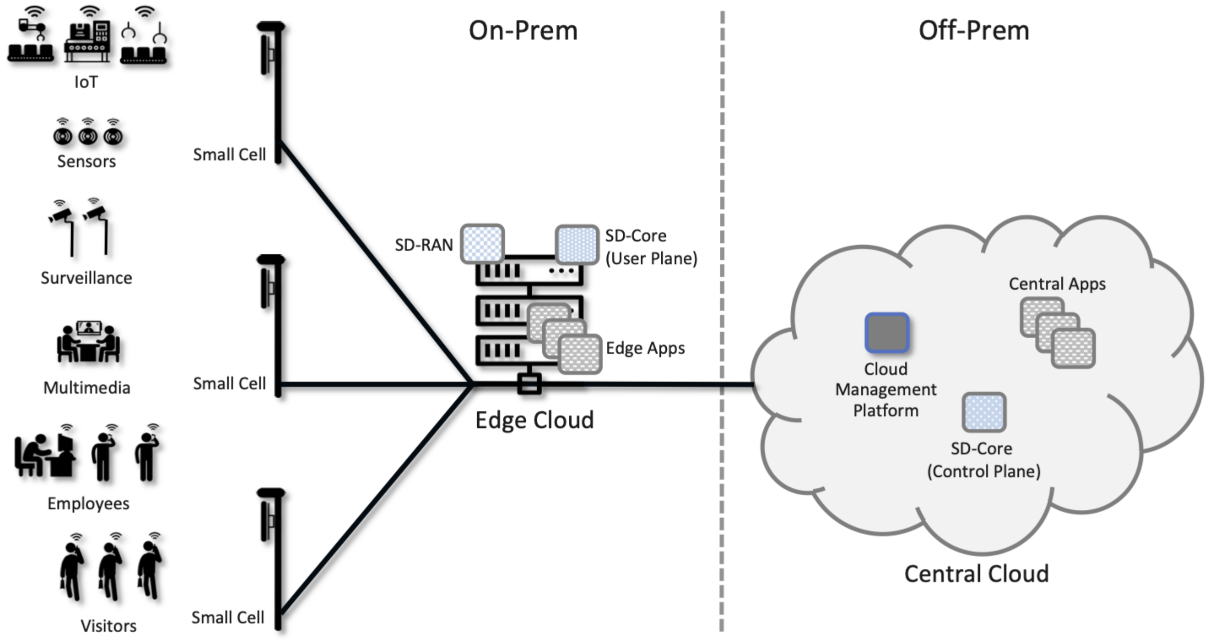

Figure 2.14.: Overview of Aether as a hybrid cloud, with edge apps and the 5G data plane (called *local breakout*) running on-prem and various management and control-related workloads running in a central cloud.

The approach includes both edge (on-prem) and centralized (off-prem) components. This is true for edge apps, which often have a centralized counterpart running in a commodity cloud. It is also true for the 5G Mobile Core, where the on-prem User Plane (UP) is paired with a centralized Control Plane (CP). The central cloud shown in this figure might be private (i.e., operated by the enterprise), public (i.e., operated by a commercial cloud provider), or some combination of the two (i.e., not all centralized elements need to run in the same cloud).

Also shown in Figure 2.14 is a centralized *Control and Management Platform*. This is Aether's version of the "Management Layer" depicted in Figure 2.13, and it represents all the functionality needed to offer Aether as a managed cloud service, with system administrators using a portal exported by this platform to operate the underlying infrastructure and services within their enterprise.

Once we deconstruct the individual components in more details in the next three chapters, we return to the question of how the resulting set of components can be assembled into an operational edge cloud in Chapter 6. The end result is 5G connectivity as a managed cloud service.

CHAPTER

THREE

CHAPTER 3: RADIO TRANSMISSION

For anyone familiar with wireless access technologies that offer a best-effort service, such as Wi-Fi, the cellular network presents a notable contrast. This is mostly because cellular networks carefully allocate available radio spectrum among many users (or more precisely, UEs), aiming to deliver a certain quality to all active users in a coverage area, while also allowing those users to remain connected while moving. They also aim to maximize the efficiency of spectrum usage, as it is a finite and often costly resource. This has resulted in a highly dynamic and adaptive approach, in which coding, modulation and scheduling play a central role. 5G takes the cellular approach to managing radio resources to a new level of sophistication.

Wireless transmission is full of challenges that don't arise in wired networks. Interference can arise from many sources ranging from microwave ovens to baby monitors, while radio signals reflect off objects such as buildings and vehicles. Some objects absorb wireless signals. The properties of the wireless channel vary over time, and depending on the frequency in use. Much of the design of cellular radio systems is motivated by the need to deal with these challenges.

As we will see in this chapter, mobile cellular networks use a reservation-based strategy, whereas Wi-Fi is contention-based. This difference is rooted in each system's fundamental assumption about utilization: Wi-Fi assumes a lightly loaded network (and hence optimistically transmits when the wireless link is idle and backs off if contention is detected), while 5G assumes (and strives for) high utilization (and hence explicitly assign different users to different "shares" of the available radio spectrum).

The fact that 5G controls spectrum allocation carefully is central to its ability to deliver guarantees of bandwidth and latency more predictably than Wi-Fi. This in turn is why there is so much interest in using 5G for mission-critical applications.

We start by giving a short primer on radio transmission as a way of laying a foundation for understanding the rest of the 5G architecture. The following is not a substitute for a theoretical treatment of the topic, but is instead intended as a way of grounding the systems-oriented description of 5G that follows in the reality of wireless communication.

3.1 3.1 Coding and Modulation

The mobile channel over which digital data needs to be reliably transmitted brings a number of impairments, including noise, attenuation, distortion, fading, and interference. This challenge is addressed by a combination of coding and modulation, as depicted in Figure 3.1.

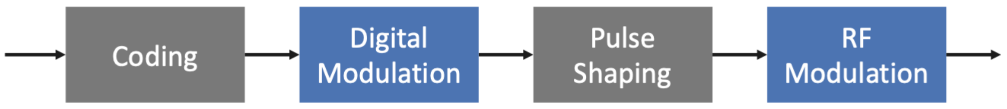

Figure 3.1.: The role of coding and modulation in mobile communication.

At its core, coding inserts extra bits into the data to help recover from all the environmental factors that interfere with signal propagation. This typically implies some form of *Forward Error Correction* (e.g., turbo codes, polar codes). Modulation then generates signals that represent the encoded data stream, and it does so in a way that matches the channel characteristics: It first uses a digital modulation signal format that maximizes the number of reliably transmitted bits every second based on the specifics of the observed channel impairments; it next matches the transmission bandwidth to channel bandwidth using pulse shaping; and finally, it uses RF modulation to transmit the signal as an electromagnetic wave over an assigned *carrier frequency.*

For a deeper appreciation of the challenges of reliably transmitting data by propagating radio signals through the air, consider the scenario depicted in Figure 3.2, where the signal bounces off various stationary and moving objects, following multiple paths from the transmitter to the receiver, who may also be moving.

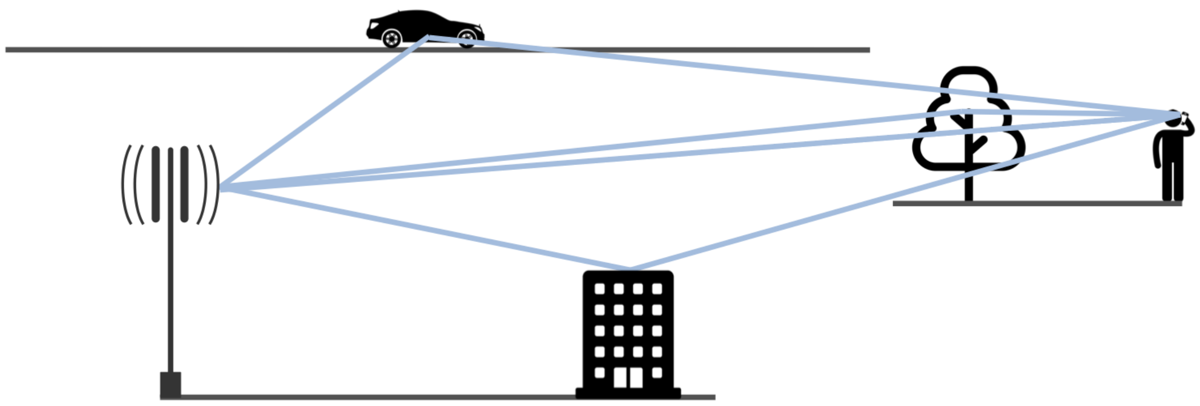

Figure 3.2.: Signals propagate along multiple paths from transmitter to receiver.

As a consequence of these multiple paths, the original signal arrives at the receiver spread over time, as illustrated in Figure 3.3. Empirical evidence shows that the Multipath Spread—the time between the first and last signals of one transmission arriving at the receiver—is 1-10μs in urban environments and 10-30μs in suburban environments. These multipath signals can interfere with each other constructively or destructively, and this will vary over time. Theoretical bounds for the time duration for which the channel may be assumed to be invariant, known as the *Coherence Time* and denoted T_c, is given by

$$T_c = c/v \times 1/f$$

where c is the velocity of the signal, v is the velocity of the receiver (e.g., moving car or train), and f is the frequency of the carrier signal that is being modulated. This says the coherence time is inversely proportional to the frequency of the signal and the speed of movement, which makes intuitive sense: The higher the frequency (narrower the wave) the shorter the coherence time, and likewise, the faster the receiver is moving the shorter the coherence time. Based on the target parameters to this model (selected according to the target physical environment), it is possible to calculate T_c, which in turn bounds the rate at which symbols can be transmitted without undue risk of interference. The dynamic nature of the wireless channel is a central challenge to address in the cellular network.

To complicate matters further, Figure 3.2 and 3.3 imply the transmission originates from a single antenna, but cell towers are equipped with an array of antennas, each transmitting in a different (but overlapping) direction. This technology, called *Multiple-Input-Multiple-Output (MIMO)*, opens the door to purposely transmitting data from multiple antennas in an effort to reach the receiver, adding even more paths to the environment-imposed multipath propagation.

One of the most important consequences of these factors is that the transmitter must receive feedback from every receiver to judge how to best utilize the wireless medium on their behalf. 3GPP specifies a

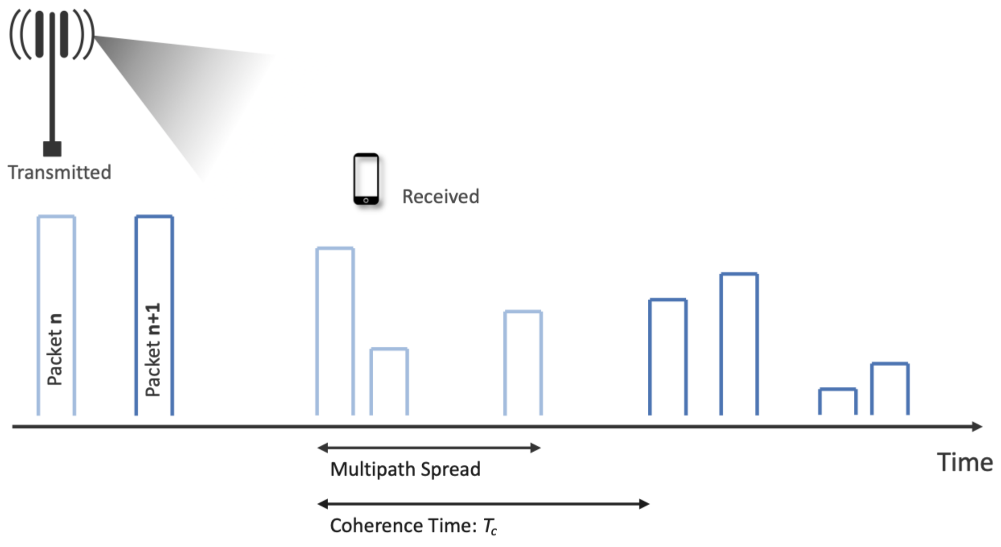

Figure 3.3.: Received data spread over time due to multipath variation.

Channel Quality Indicator (CQI) for this purpose. In practice, the receiver sends a CQI status report to the base station periodically (e.g., every millisecond). These CQI messages report the observed signal-to-noise ratio, which impacts the receiver's ability to recover the data bits. The base station then uses this information to adapt how it allocates the available radio spectrum to the subscribers it is serving, as well as which coding and modulation scheme to employ. All of these decisions are made by the scheduler.

3.2 3.2 Scheduler

How the scheduler does its job is one of the most important properties of each generation of the cellular network, which in turn depends on the multiplexing mechanism. For example, 2G used *Time Division Multiple Access (TDMA)* and 3G used *Code Division Multiple Access (CDMA)*. How data is multiplexed is also a major differentiator for 4G and 5G, completing the transition from the cellular network being fundamentally circuit-switched to fundamentally packet-switched.

Both 4G and 5G are based on *Orthogonal Frequency-Division Multiplexing (OFDM)*, an approach that multiplexes data over multiple orthogonal subcarrier frequencies, each of which is modulated independently. One attraction of OFDM is that, by splitting the frequency band into subcarriers, it can send symbols on each subcarrier at a relatively low rate. This makes it easier to correctly decode symbols in the presence of multipath interference. The efficiency of OFDM depends on the selection of subcarrier frequencies so as to avoid interference, that is, how it achieves orthogonality. That topic is beyond the scope of this book.

As long as you understand that OFDM uses a set of subcarriers, with symbols (each of which contain a few bits of data) being sent at some rate on each subcarrier, that you can appreciate that there are discrete schedulable units of the radio spectrum. The fundamental unit is the time to transmit one symbol on one subcarrier. With that building block in mind, we are now in a position to examine how multiplexing and scheduling work in 4G and 5G.

3.2.1 3.2.1 Multiplexing in 4G

We start with 4G because it provides a foundational understanding that makes 5G easier to explain, where both 4G and 5G use an approach to multiplexing called *Orthogonal Frequency-Division Multiple Access (OFDMA)*. You can think of OFDMA as a specific application of OFDM. In the 4G case, OFDMA multiplexes data over a set of 12 orthogonal (non-interfering) subcarrier frequencies, each of which is modulated independently.[1] The "Multiple Access" in OFDMA implies that data can simultaneously be sent on behalf of multiple users, each on a different subcarrier frequency and for a different duration of time. The 4G-defined subbands are narrow (e.g., 15 kHz), and the coding of user data into OFDMA symbols is designed to minimize the risk of data loss due to interference.

The use of OFDMA naturally leads to conceptualizing the radio spectrum as a 2-D resource, as shown in Figure 3.4, with the subcarriers represented in the vertical dimension and the time to transmit symbols on each subcarrier represented in the horizontal dimension. The basic unit of transmission, called a *Resource Element (RE)*, corresponds to a 15-kHz band around one subcarrier frequency and the time it takes to transmit one OFDMA symbol. The number of bits that can be encoded in each symbol depends on the modulation scheme in use. For example, using *Quadrature Amplitude Modulation (QAM)*, 16-QAM yields 4 bits per symbol and 64-QAM yields 6 bits per symbol. The details of the modulation need not concern us here; the key point is that there is a degree of freedom to choose the modulation scheme based on the measured channel quality, sending more bits per symbol (and thus more bits per second) when the quality is high.

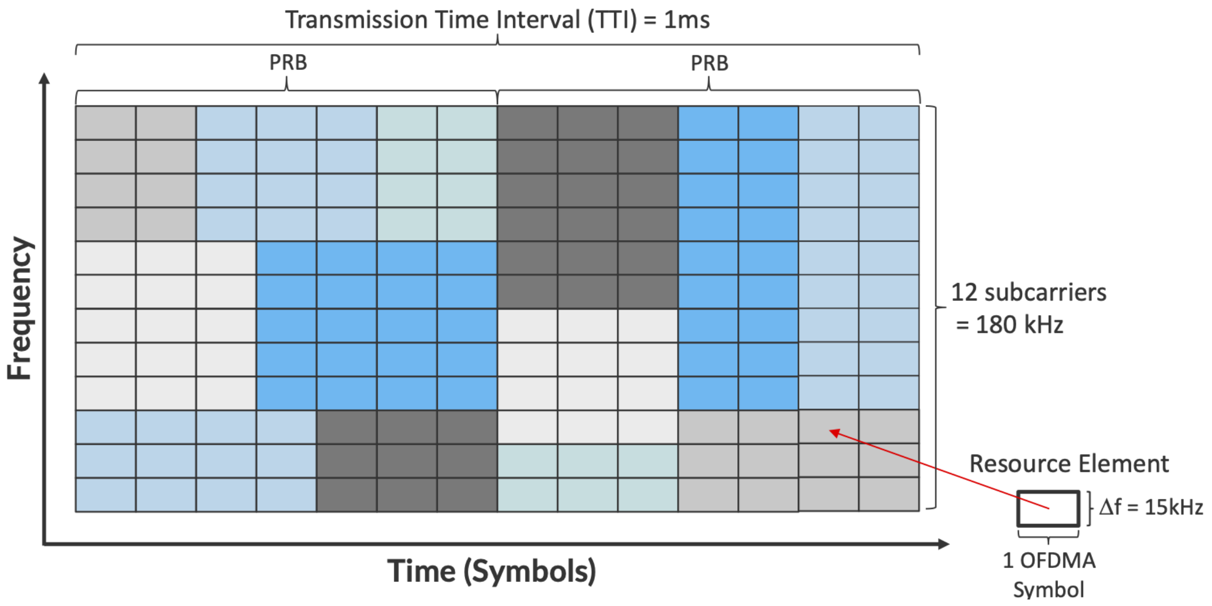

Figure 3.4.: Spectrum abstractly represented by a 2-D grid of schedulable Resource Elements.

A scheduler allocates some number of REs to each user that has data to transmit during each 1 ms *Transmission Time Interval (TTI)*, where users are depicted by different colored blocks in Figure 3.4. The only constraint on the scheduler is that it must make its allocation decisions on blocks of 7x12=84 resource elements, called a *Physical Resource Block (PRB)*. Figure 3.4 shows two back-to-back PRBs. Of course time continues to flow along one axis, and depending on the size of the available frequency band (e.g., it might be 100 MHz wide), there may be many more subcarrier slots (and hence PRBs) available along the other axis, so the scheduler is essentially preparing and transmitting a sequence of PRBs.

Note that OFDMA is not a coding/modulation algorithm, but instead provides a framework for selecting a

[1] 4G uses OFDMA for downstream traffic, and a different multiplexing strategy for upstream transmissions (from user devices to base stations), but we do not describe it because the approach is not applicable to 5G.

specific coding and modulation for each subcarrier frequency. QAM is one common example modulation. It is the scheduler's responsibility to select the modulation to use for each PRB, based on the CQI feedback it has received. The scheduler also selects the coding on a per-PRB basis, for example, by how it sets the parameters to the turbo code algorithm.

The 1-ms TTI corresponds to the time frame in which the scheduler receives feedback from users about the quality of the signal they are experiencing. This is the role of CQI: once every millisecond, each user sends a set of metrics, which the scheduler uses to make its decision as to how to allocate PRBs during the subsequent TTI.

Another input to the scheduling decision is the *QoS Class Identifier (QCI)*, which indicates the quality-of-service each class of traffic is to receive. In 4G, the QCI value assigned to each class (there are twenty six such classes, in total) indicates whether the traffic has a *Guaranteed Bit Rate (GBR)* or not *(non-GBR)*, plus the class's relative priority within those two categories. (Note that the 5QI parameter introduced in Chapter 2 serves the same purpose as the QCI parameter in 4G.)

Finally, keep in mind that Figure 3.4 focuses on scheduling transmissions from a single antenna, but the MIMO technology described above means the scheduler also has to determine which antenna (or more generally, what subset of antennas) will most effectively reach each receiver. But again, in the abstract, the scheduler is charged with allocating a sequence of Resource Elements.

Note that the scheduler has many degrees of freedom: it has to decide which set of users to service during a given time interval, how many resource elements to allocate to each such user, how to select the coding and modulation levels, and which antenna to transmit their data on. This is an optimization problem that, fortunately, we are not trying to solve here. Our goal is to describe an architecture that allows someone else to design and plug in an effective scheduler. Keeping the cellular architecture open to innovations like this is one of our goals, and as we will see in the next section, becomes even more important in 5G where the scheduler operates with even more degrees of freedom.

3.2.2 3.2.2 Multiplexing in 5G

The transition from 4G to 5G introduces additional flexibility in how the radio spectrum is scheduled, making it possible to adapt the cellular network to a more diverse set of devices and application domains.

Fundamentally, 5G defines a family of waveforms—unlike LTE, which specified only one waveform—each optimized for a different band in the radio spectrum.[2] The bands with carrier frequencies below 1 GHz are designed to deliver mobile broadband and massive IoT services with a primary focus on range. Carrier frequencies between 1-6 GHz are designed to offer wider bandwidths, focusing on mobile broadband and mission-critical applications. Carrier frequencies above 24 GHz (mmWaves) are designed to provide super-wide bandwidths over short, line-of-sight coverage.

These different waveforms affect the scheduling and subcarrier intervals (i.e., the "size" of the resource elements described in the previous section).

- For frequency range 1 (410 MHz - 7125 MHz), 5G allows maximum 100 MHz bandwidths. In this case, there are three waveforms with subcarrier spacings of 15, 30 and 60 kHz. (We used 15 kHz in the example shown in Figure 3.4.) The corresponding to scheduling intervals of 0.5, 0.25, and 0.125 ms, respectively. (We used 0.5 ms in the example shown in Figure 3.4.)
- For millimeter bands, also known as frequency range 2 (24.25 GHz - 52.6 GHz), bandwidths may go from 50 MHz up to 400 MHz. There are two waveforms, with subcarrier spacings of 60 kHz and 120 kHz. Both have scheduling intervals of 0.125 ms.

[2] A waveform is defined by the frequency, amplitude, and phase-shift independent property (shape) of a signal. A sine wave is a simple example of a waveform.

These various configurations of subcarrier spacing and scheduling intervals are sometimes called the *numerology* of the radio's air interface.

This range of numerology is important because it adds another degree of freedom to the scheduler. In addition to allocating radio resources to users, it has the ability to dynamically adjust the size of the resource by changing the waveform being used. With this additional freedom, fixed-sized REs are no longer the primary unit of resource allocation. We instead use more abstract terminology, and talk about allocating *Resource Blocks* to subscribers, where the 5G scheduler determines both the size and number of Resource Blocks allocated during each time interval.

Figure 3.5 depicts the role of the scheduler from this more abstract perspective. Just as with 4G, CQI feedback from the receivers and the 5QI quality-of-service class selected by the subscriber are the two key pieces of input to the scheduler. Note that the set of 5QI values available in 5G is considerably greater than its QCI counterpart in 4G, reflecting the increasing differentiation among classes that 5G aims to support. For 5G, each class includes the following attributes:

- Resource Type: Guaranteed Bit Rate (GBR), Delay-Critical GBR, Non-GBR
- Priority Level
- Packet Delay Budget
- Packet Error Rate
- Maximum Data Burst
- Averaging Window

Note that while the preceding discussion could be interpreted to imply a one-to-one relationship between subscribers and a 5QI, it is more accurate to say that each 5QI is associated with a class of traffic (often corresponding to some type of application), where a given subscriber might be sending and receiving traffic that belongs to multiple classes at any given time.

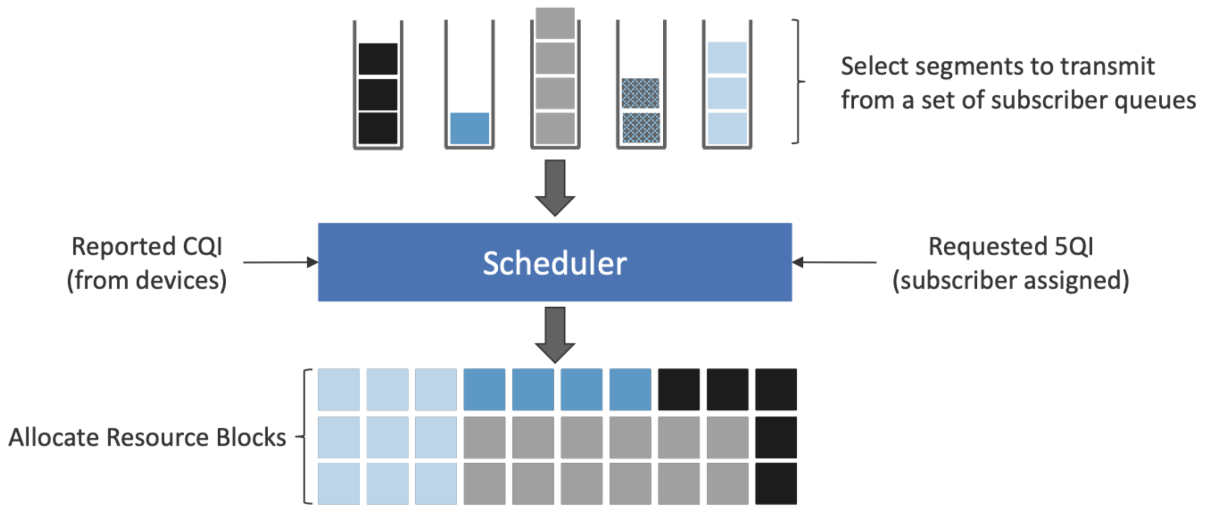
Select segments to transmit from a set of subscriber queues
Reported CQI (from devices)
Scheduler
Requested 5QI (subscriber assigned)
Allocate Resource Blocks

Figure 3.5.: Scheduler allocates Resource Blocks to user data streams based on CQI feedback from receivers and the 5QI parameters associated with each class of service.

3.3 3.3 Virtualized Scheduler (Slicing)

The discussion up to this point presumes a single scheduler is suitable for all workloads, but different applications have different requirements for how their traffic gets scheduled. For example, some applications care about latency and others care more about bandwidth.

While in principle one could define a sophisticated scheduler that takes dozens of different factors into account, 5G has introduced a mechanism that allows the underlying resources (in this case radio spectrum) to be "sliced" between different uses. The key to slicing is to add a layer of indirection between the scheduler and the physical resource blocks. Slicing, like much of 5G, has received a degree of hype, but it boils down to virtualization at the level of the radio scheduler.

As shown in Figure 3.6, the idea is to first schedule offered traffic demands to virtual RBs, and then perform a second mapping of Virtual RBs to Physical RBs. This sort of virtualization is common in resource allocators throughout computing systems because we want to separate how many resources are allocated to each user (or virtual machine in the computing case) from the decision as to which physical resources are actually assigned. This virtual-to-physical mapping is performed by a layer typically known as a *Hypervisor*, and the important thing to keep in mind is that it is totally agnostic about which user's segment is affected by each translation.

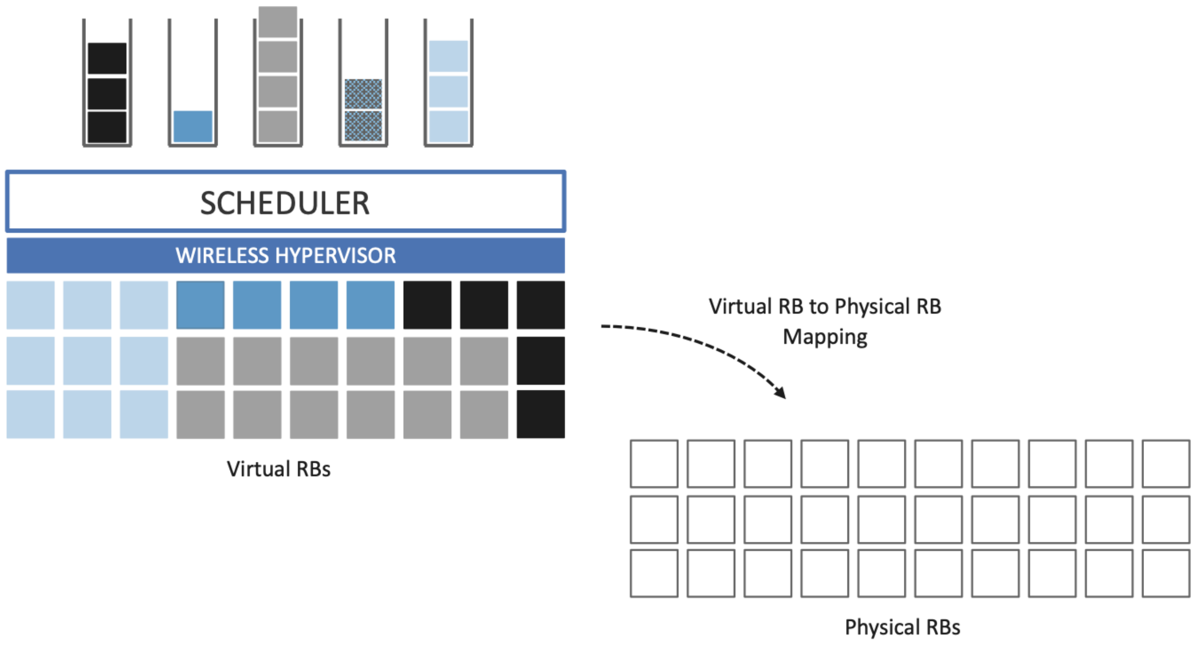

Figure 3.6.: Wireless Hypervisor mapping virtual resource blocks to physical resource blocks.

Having decoupled the Virtual RBs from Physical RBs, it is now possible to define multiple Virtual RB sets (of varying sizes), each with its own scheduler. Figure 3.7 gives an example with two equal-sized RB sets. The important consequence is this: having made the macro-decision that the Physical RBs are divided into two equal partitions, the scheduler associated with each partition is free to allocate Virtual RBs independently from the other. For example, one scheduler might be designed to deal with high-bandwidth video traffic and another scheduler might be optimized for low-latency IoT traffic. Alternatively, a certain fraction of the available capacity could be reserved for premium customers or other high-priority traffic (e.g., public safety), with the rest shared among everyone else.

A final point to note is that there is considerable flexibility in the allocation of resources to slices. While the example above shows resources allocated in a fixed manner to each slice, it is possible to make unused resources in one slice available to another slice, as long as they can be reclaimed when needed. This is

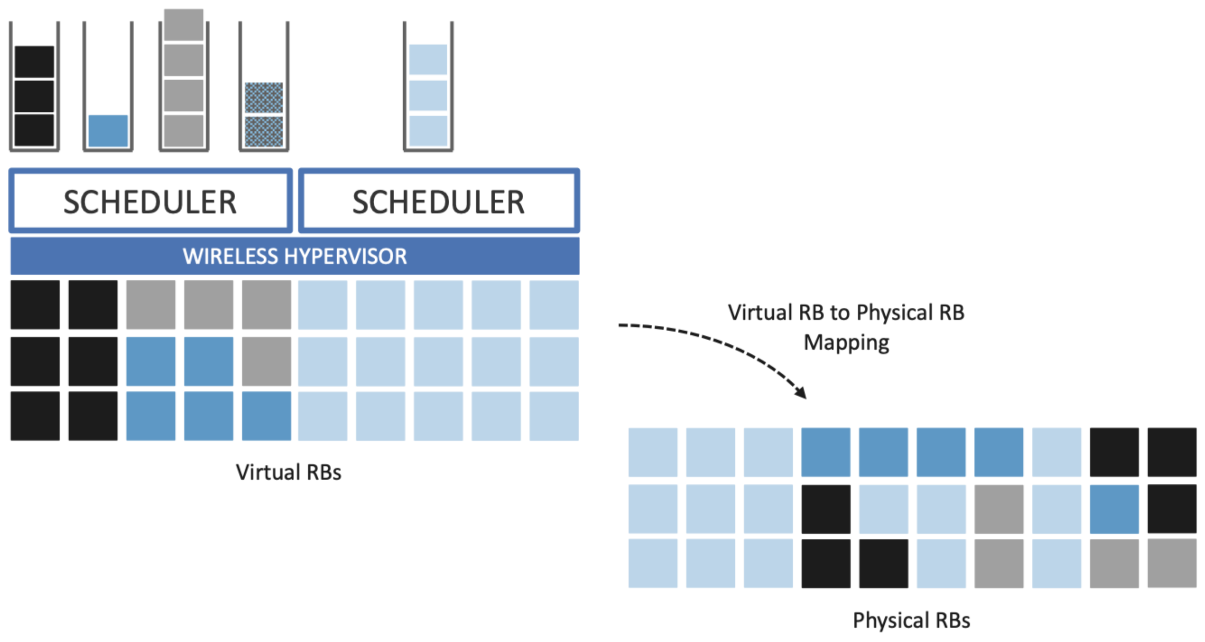

Figure 3.7.: Multiple schedulers running on top of wireless hypervisor.

similar to how work-conserving scheduling works in network queues: resources are not wasted if the offered load in a class is less than the rate guaranteed for that class.

3.4 3.4 New Use Cases

We conclude by noting that up to this point we have described 5G as introducing additional degrees of freedom into how data is scheduled for transmission, but when taken as a whole, the end result is a qualitatively more powerful radio. This new 5G air interface specification, which is commonly referred to as *New Radio (NR)*, enables new use cases that go well beyond simply delivering increased bandwidth. 3GPP defined five such use cases:

- Enhanced Mobile Broadband (eMBB)
- Ultra-Reliable Low-Latency Communications (URLLC)
- Massive Internet of Things (MIoT)
- Vehicle to Infrastructure/Vehicle (V2X)
- High-Performance Machine-Type Communications (HMTC)

These use cases reflect the requirements introduced in Chapter 1, and can be attributed to four fundamental improvements in how 5G multiplexes data onto the radio spectrum.

The first improvement is being able to change the waveform. This effectively introduces the ability to dynamically change the size and number of schedulable resource units, which opens the door to making fine-grained scheduling decisions that are critical to predictable, low-latency communication.

The second is related to the “Multiple Access” aspect of how distinct traffic sources are multiplexed onto the available spectrum. In 4G, multiplexing happens in both the frequency and time domains for downstream traffic, but only in the frequency domain for upstream traffic. 5G NR multiplexes both upstream and downstream traffic in both the time and frequency domains. Doing so provides finer-grained scheduling control needed by latency-sensitive applications.

The third advance is related to the new spectrum available to 5G NR, with the mmWave allocations opening above 24 GHz being especially important. This is not only because of the abundance of capacity—which makes it possible to set aside dedicated capacity for applications that require low-latency communication—but also because the higher frequency enables even finer-grained resource blocks (e.g., scheduling intervals as short as 0.125 ms). Again, this improves scheduling granularity to the benefit of applications that cannot tolerate unpredictable latency.

The fourth is related to delivering mobile connectivity to a massive number of IoT devices, ranging from devices that require mobility support and modest data rates (e.g. wearables, asset trackers) to devices that support intermittent transmission of a few bytes of data (e.g., sensors, meters). None of these devices are particularly latency-sensitive or bandwidth-hungry, but they often require long battery lifetimes, and hence, reduced hardware complexity that draws less power.

Support for IoT device connectivity revolves around allocating some of the available radio spectrum to a light-weight (simplified) air interface. This approach started with Release 13 of LTE via two complementary technologies: mMTC and NB-IoT (NarrowBand-IoT). Both technologies build on a significantly simplified version of LTE—i.e., limiting the numerology and flexibility needed to achieve high spectrum utilization—so as to allow for simpler IoT hardware design. mMTC delivers up to 1 Mbps over 1.4 MHz of bandwidth and NB-IoT delivers a few tens of kbps over 200 kHz of bandwidth; hence the term *NarrowBand*. Both technologies have been designed to support over 1 million devices per square kilometer. With Release 16, both technologies can be operated in-band with 5G, but still based on LTE numerology. Starting with Release 17, a simpler version of 5G NR, called *NR-Light*, will be introduced as the evolution of mMTC. NR-Light is expected to scale the device density even further.

As a complement to these four improvements, 5G NR is designed to support partitioning the available bandwidth, with different partitions dynamically allocated to different classes of traffic (e.g., high-bandwidth, low-latency, and low-complexity). This is the essence of *slicing*, as discussed above. Moreover, once traffic with different requirements can be served by different slices, 5G NR's approach to multiplexing is general enough to support varied scheduling decisions for those slices, each tailored for the target traffic. We return to the applications of slicing in Chapter 6.

CHAPTER

FOUR

CHAPTER 4: RADIO ACCESS NETWORK

The high-level description of the RAN in Chapter 2 was mostly silent about the RAN's internal structure. We now focus on those details, and in doing so, explain how the RAN is being transformed in 5G.

You can think of the RAN as having one main job: to transfer packets between the mobile core and a set of UEs. This means it is deeply involved in the management and scheduling of radio spectrum that we discussed in the last chapter, but there is more to it than that. We start by describing the stages in the RAN's packet processing pipeline, and then showing how these stages are being disaggregated, distributed, and implemented.

Note that the deconstruction of the RAN presented in this chapter represents a combination of standardized specifications and implementation strategies. The former continues to be under the purview of the 3GPP, but the latter are primarily influenced by a second organization: the *Open-RAN Alliance (O-RAN)* introduced in Chapter 1. O-RAN is led by network operators with the goal of developing a software-based implementation of the RAN that eliminates vendor lock-in. Such business forces are certainly a factor in where 5G mobile networks are headed, but our goal in this chapter is to identify the technical design decisions involved in that evolution.

4.1 4.1 Packet Processing Pipeline

Figure 4.1 shows the packet processing stages historically bundled in base stations, as specified by the 3GPP standard. Note that the figure depicts the base station as a pipeline (running left-to-right for packets sent to the UE) but it is equally valid to view it as a protocol stack (as is typically done in official 3GPP documents). Also note that (for now) we are agnostic as to how these stages are implemented. Since we are ultimately heading towards a cloud-based implementation, one possible implementation strategy would be a microservice per box.

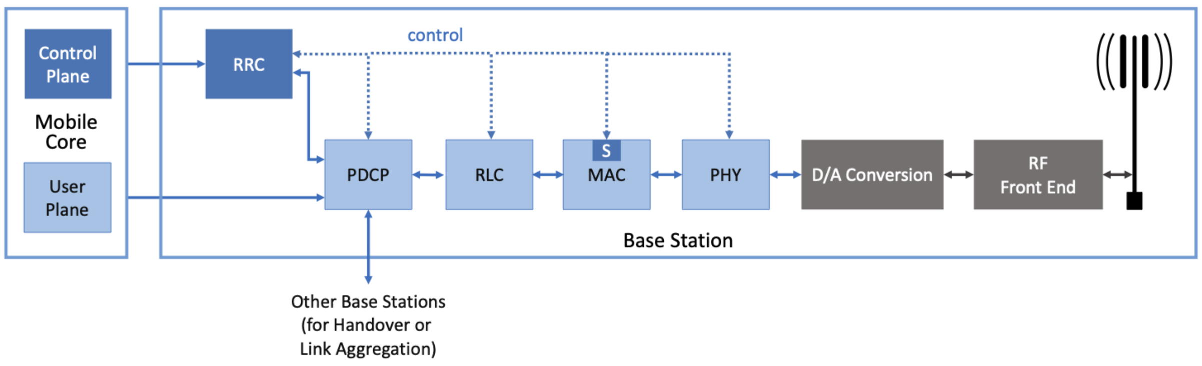

Figure 4.1.: RAN processing pipeline, including both user and control plane components.

The key stages are as follows.

- RRC (Radio Resource Control) → Responsible for configuring the coarse-grained and policy-related aspects of the pipeline. The RRC runs in the RAN's control plane; it does not process packets in the user plane.
- PDCP (Packet Data Convergence Protocol) → Responsible for compressing and decompressing IP headers, ciphering and deciphering, integrity protection and integrity verification, duplication, reordering and in-order delivery, and out-of-order delivery.
- RLC (Radio Link Control) → Responsible for segmentation and reassembly, as well as reliably transmitting/receiving segments using error correction through ARQ (automatic repeat request).
- MAC (Media Access Control) → Responsible for buffering, multiplexing and demultiplexing segments, including all real-time scheduling decisions about what segments are transmitted when. Also able to make a "late" forwarding decision (i.e., to alternative carrier frequencies, including Wi-Fi).
- PHY (Physical Layer) → Responsible for coding and modulation (as discussed in Chapter 3), including FEC.

The last two stages in Figure 4.1 (D/A conversion and the RF front-end) are beyond the scope of this book.

While it is simplest to view the stages in Figure 4.1 as a pure left-to-right pipeline, the Scheduler described in Section 3.2 (denoted "S" in the figure) runs in the MAC stage/layer, and implements the "main loop" for outbound traffic: It reads data from the upstream RLC and schedules transmissions to the downstream PHY. Since the Scheduler determines the number of bytes to transmit to a given UE during each time period (based on all the factors outlined in Chapter 3), it must request (get) a segment of that length from the upstream queue. In practice, the size of the segment that can be transmitted on behalf of a single UE during a single scheduling interval can range from a few bytes to an entire IP packet.

Also note that a combination of the RRC and PDCP are responsible for the observation made in Section 2.3: that a *"base station can be viewed as a specialized forwarder"*. The control plane logic that decides whether this base station should continue processing a packet or forward it to another base station runs in the RRC, and the corresponding user plane mechanism that carries out the forwarding decision is implemented in the PDCP. The interface between these two elements is defined by the 3GPP spec, but the decision-making logic is an implementation choice (and historically proprietary). This control logic is generically referred as the *Radio Resource Management (RRM)*, not to be confused with the standards-defined RRC stage depicted in Figure 4.1.

4.2 4.2 Split RAN

The next step is to understand how the functionality outlined above is partitioned between physical elements, and hence, "split" across centralized and distributed locations. The dominant option has historically been "no split," with the entire pipeline shown in Figure 4.1 running in the base station. Going forward, the 3GPP standard has been extended to allow for multiple split-points, with the partition shown in Figure 4.2 being actively pursued by the operator-led O-RAN Alliance. It is the split we adopt throughout the rest of this book. Note that the split between centralized and distributed components mirrors the split made in SDN, with similar motivations. We discuss further how SDN techniques are applied to the RAN below.

This results in a RAN-wide configuration similar to that shown in Figure 4.3, where a single *Central Unit (CU)* running in the cloud serves multiple *Distributed Units (DUs)*, each of which in turn serves multiple *Radio Units (RUs)*. Critically, the RRC (centralized in the CU) is responsible for making only

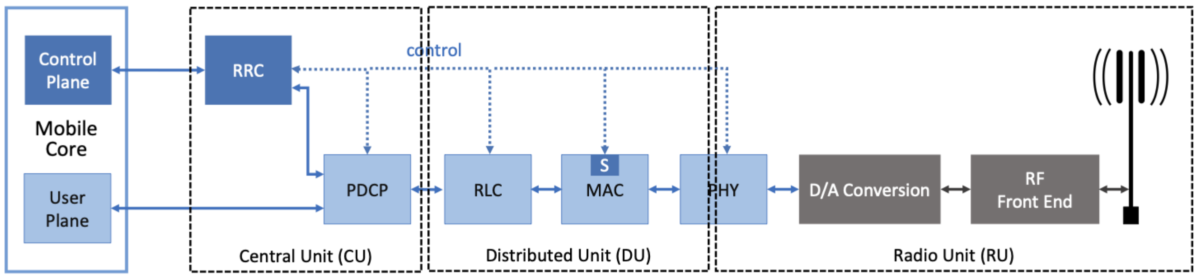

Figure 4.2.: Split RAN processing pipeline distributed across a Central Unit (CU), Distributed Unit (DU), and Radio Unit (RU).

near-real-time configuration and control decisions, while the Scheduler that is part of the MAC stage is responsible for all real-time scheduling decisions.

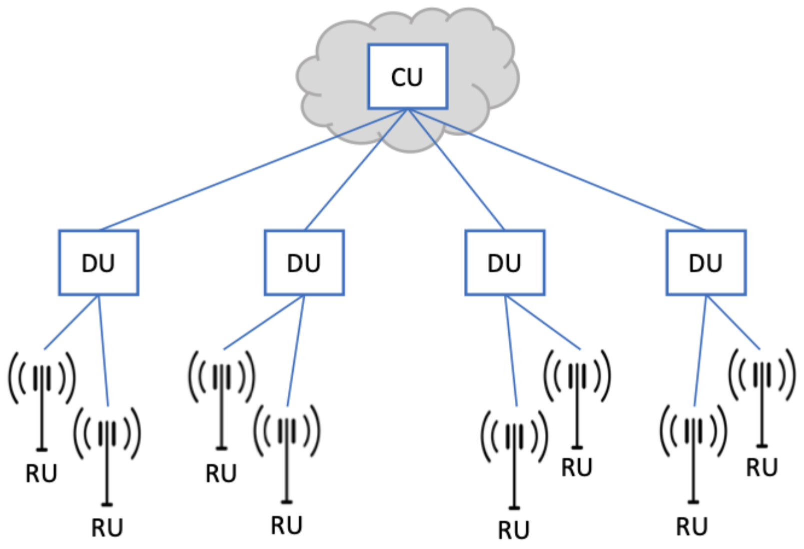

Figure 4.3.: Split RAN hierarchy, with one CU serving multiple DUs, each of which serves multiple RUs.

Because scheduling decisions for radio transmission are made by the MAC layer in real time, a DU needs to be "near" (within 1ms) the RUs it manages. (You can't afford to make scheduling decisions based on out-of-date channel information.) One familiar configuration is to co-locate a DU and an RU in a cell tower. But when an RU corresponds to a small cell, many of which might be spread across a modestly-sized geographic area (e.g., a mall, campus, or factory), then a single DU would likely service multiple RUs. The use of mmWave in 5G is likely to make this latter configuration all the more common.

Also note that the Split RAN changes the nature of the Backhaul Network, which originally connected the base stations back to the Mobile Core. With the Split RAN there are multiple connections, which are officially labeled as follows.

- RU-DU connectivity is called the Fronthaul
- DU-CU connectivity is called the Midhaul
- CU-Mobile Core connectivity is called the Backhaul

For more insight into design considerations for interconnecting the distributed components of a Split RAN, we recommend the NGMN Alliance Report.

Further Reading

RAN Evolution Project: Backhaul and Fronthaul Evolution. NGMN Alliance Report, March 2015.

One observation about the CU (which becomes relevant in Chapter 6 when we incorporate it into a managed cloud service) is that one might co-locate the CU and Mobile Core in the same cluster, meaning the backhaul is implemented in the cluster switching fabric. In such a configuration, the midhaul then effectively serves the same purpose as the original backhaul, and the fronthaul is constrained by the predictable/low-latency requirements of the MAC stage's real-time scheduler. This situation is further complicated by the fact that the mobile core itself may be disaggregated, as discussed in Chapter 5.

A second observation about the CU shown in Figure 4.2 is that it encompasses two functional blocks—the RRC and the PDCP—which lie on the RAN's control plane and user plane, respectively. This separation is consistent with the idea of CUPS introduced in Chapter 2, and plays an increasingly important role as we dig deeper into how the RAN is implemented. For now, we note that the two parts are sometimes referred to as the CU-CP and CU-UP, respectively.

We conclude our description of the split RAN architecture with the alternative depiction in Figure 4.4. For completeness, this figure identifies the standardized interfaces between the components (e.g., N2, N3, F1-U, F1-C, and Open Fronthaul). We're not going to talk about these interfaces, except to note that they exist and there is a corresponding 3GPP specification that spells out the details. Instead, we're going to comment on the availability of open source implementations for each component.

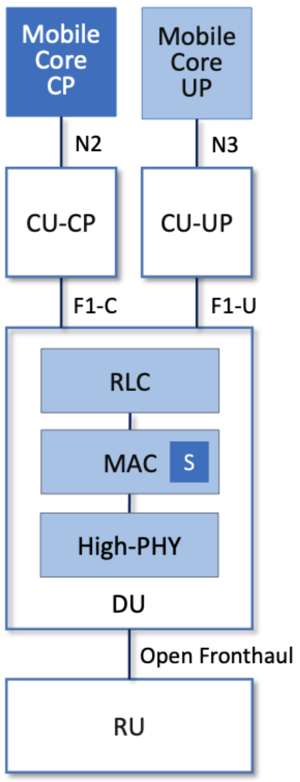

Figure 4.4.: Alternative depiction of the Split RAN components, showing the 3GPP-specified inter-unit interfaces.

With respect to the Central Unit, most of the complexity is in the CU-CP, which, as we'll see in the next section, is being re-engineered using SDN, with open source solutions available. With respect to the Radio Unit, nearly all the complexity is in D/A conversion and how the resulting analog signal is amplified. Incumbent vendors have significant proprietary know-how in this space, which will almost certainly remain closed.

With respect to the Distributed Unit, the news is mixed, and correspondingly, the figure shows more detail. The High-PHY module—which corresponds to all but the RF modulation step of Figure 3.1 in Section 3.1—is one of the most complex components in the RAN stack. An open source implementation of the High-PHY, known as FlexRAN, exists and is widely used in commercial products. The only caveat is that the software license restricts usage to Intel processors, although it is also the case that the FlexRAN software exploits Intel-specific hardware capabilities. As for the rest of the DU, the MAC is the other source of high-value closed technology, particularly in how scheduling is done. There is an open source version made available by the Open Air Initiative (OAI), but its usage is restricted to research-only deployments.

Further Reading

FlexRAN: Reference Architecture for Wireless Access.

Open Air Interface.

4.3 4.3 Software-Defined RAN

We now describe how the RAN is being implemented according to SDN principles, resulting in an SD-RAN. The key architectural insight is shown in Figure 4.5, where the RRC from Figure 4.1 is partitioned into two sub-components: the one on the left provides a 3GPP-compliant way for the RAN to interface to the Mobile Core's control plane (the figure labels this sub-component as a "Proxy"), while the one on the right opens a new programmatic API for exerting software-based control over the pipeline that implements the RAN user plane.

To be more specific, the left sub-component simply forwards control packets between the Mobile Core and the PDCP, providing a path over which the Mobile Core can communicate with the UE for control purposes, whereas the right sub-component implements the core of the RRC's control functionality (which as we explained in Section 4.1 is also known as RRM). This latter component is commonly referred to as the *RAN Intelligent Controller (RIC)* in O-RAN architecture documents, so we adopt this terminology. The "Near-Real Time" qualifier indicates the RIC is part of 10-100ms control loop implemented in the CU, as opposed to the ~1ms control loop required by the MAC scheduler running in the DU.

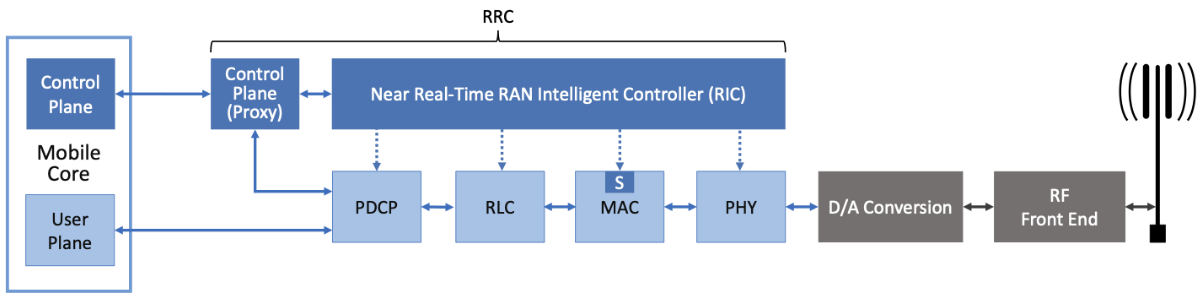

Figure 4.5.: RRC disaggregated into a Mobile Core facing control plane component (a proxy) and a Near-Real-Time Controller.

Although not shown in Figure 4.5, keep in mind (from Figure 4.2) that the RRC and the PDCP form the CU. Reconciling these two figures is a little bit messy, but to a first approximation, the PDCP corresponds to the CU-UP and RRC-Proxy corresponds to the CU-CP, with the RIC "lifted out" and responsible for overseeing both. We postpone a diagram depicting this relationship until Section 4.5, where we summarize the end-to-end result. For now, the important takeaway is that the SDN-inspired refactoring of the RAN is free both to move functionality around and to introduce new module boundaries, as long as the original 3GPP-defined interfaces are preserved.

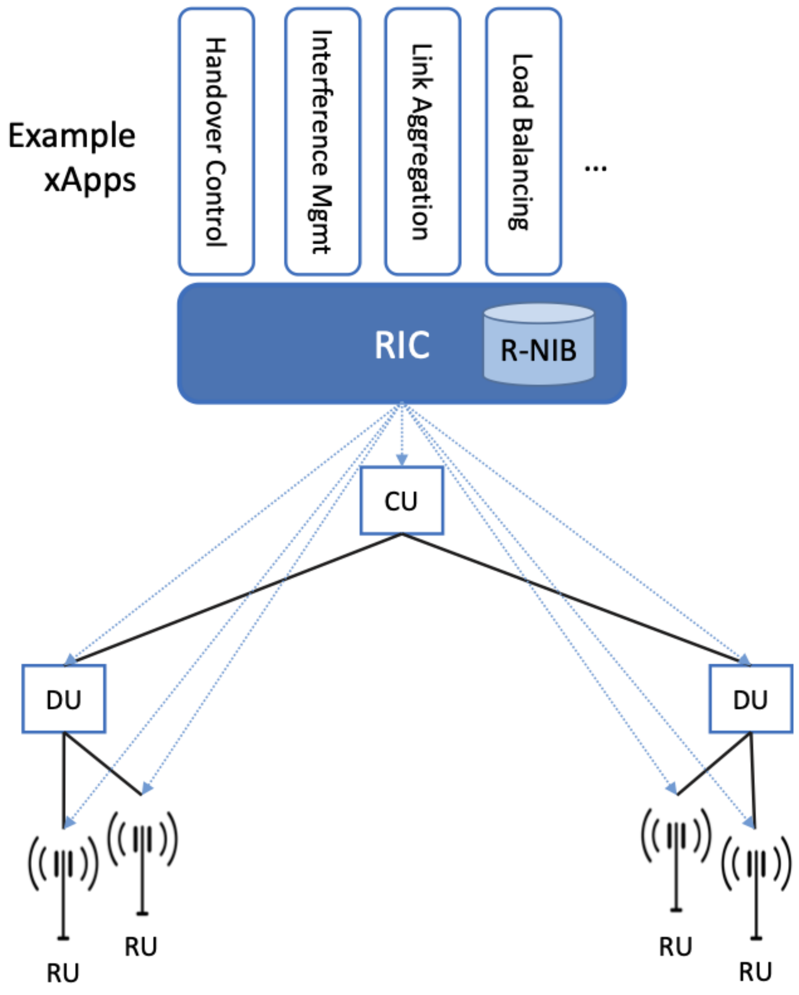

Figure 4.6.: Example set of control applications (xApps) running on top of Near-Real-Time RAN Controller (RIC), controlling a distributed set of Split RAN elements (CU, DU, RU).

Completing the picture, Figure 4.6 shows the Near-RT RIC implemented as an SDN Controller hosting a set of SDN control apps. The RIC maintains a *RAN Network Information Base (R-NIB)*—a common set of information that can be consumed by numerous control apps. The R-NIB includes time-averaged CQI values and other per-session state (e.g., GTP tunnel IDs, 5QI values for the type of traffic), while the MAC (as part of the DU) maintains the instantaneous CQI values required by the real-time scheduler. More generally, the R-NIB includes the following state:

- Fixed Nodes (RU/DU/CU Attributes)
  - Identifiers
  - Version
  - Config Report
  - RRM config
  - PHY resource usage

- Mobile Nodes (UE Attributes)
  - Devices
    - Identifiers
    - Capability
    - Measurement Config
    - State (Active/Idle)
  - Links (*Actual* and *Potential*)
    - Identifiers
    - Link Type
    - Config/Bearer Parameters
    - 5QI Value
- Virtual Constructs (Slice Attributes)
  - Links
  - Bearers/Flows
  - Validity Period
  - Desired KPIs
  - MAC RRM Configuration
  - RRM Control Configuration

The four example Control Apps (xApps) in Figure 4.6 do not constitute an exhaustive list, but they do represent the sweet spot for SDN, with its emphasis on central control over distributed forwarding. These functions—Link Aggregation Control, Interference Management, Load Balancing, and Handover Control—are often implemented by individual base stations with only local visibility, but they have global consequences. The SDN approach is to collect the available input data centrally, make a globally optimal decision, and then push the respective control parameters back to the base stations for execution. Evidence using an analogous approach to optimize wide-area networks over many years (see, for example, B4) is compelling.

Further Reading

For an example of how SDN principles have been successfully applied to a production network, we recommend B4: Experience with a Globally-Deployed Software Defined WAN. ACM SIGCOMM, August 2013.

One way to characterize xApps is based on the current practice of controlling the mobile link at two different levels. At a fine-grained level, per-node and per-link control are conducted using the RRM functions that are distributed across the individual base stations.[1] RRM functions include scheduling, handover control, link and carrier aggregation control, bearer control, and access control. At a coarse-grained level, regional mobile network optimization and configuration is conducted using *Self-Organizing Network (SON)* functions. These functions oversee neighbor lists, manage load balancing, optimize coverage

[1] Pedantically, Radio Resource Management (RRM) is another name for the collection of control functionality typically implemented in the RRC stage of the RAN pipeline.

and capacity, aim for network-wide interference mitigation, manage centrally configured parameters, and so on. As a consequence of these two levels of control, it is not uncommon to see reference to *RRM Applications* and *SON Applications*, respectively, in O-RAN documents for SD-RAN. For example, the Interference Management and Load Balancing xApps in Figure 4.6 are SON Applications, while the other two xApps are RRM Applications.

Keep in mind, however, that this characterization of xApps is based on past (pre-SDN) implementations of the RAN. This is helpful as the industry transitions to SD-RAN, but the situation is likely to change. SDN is transforming the RAN, so new ways of controlling the RAN—resulting in applications that do not fit neatly into the RRM vs SON dichotomy—can be expected to emerge over time.

4.4 4.4 Near Real-Time RIC

Drilling down to the next level of detail, Figure 4.7 shows an exemplar implementation of a RIC based on a retargeting of the Open Network OS (ONOS) for the SD-RAN use case. ONOS (described in our SDN book) was originally designed to support traditional wireline network switches using standard interfaces (OpenFlow, P4Runtime, etc.). For the SD-RAN use case, the ONOS-based RIC instead supports a set of RAN-specific north- and south-facing interfaces, but internally takes advantage of the same collection of subsystems (microservices) as in the wireline case.[2]

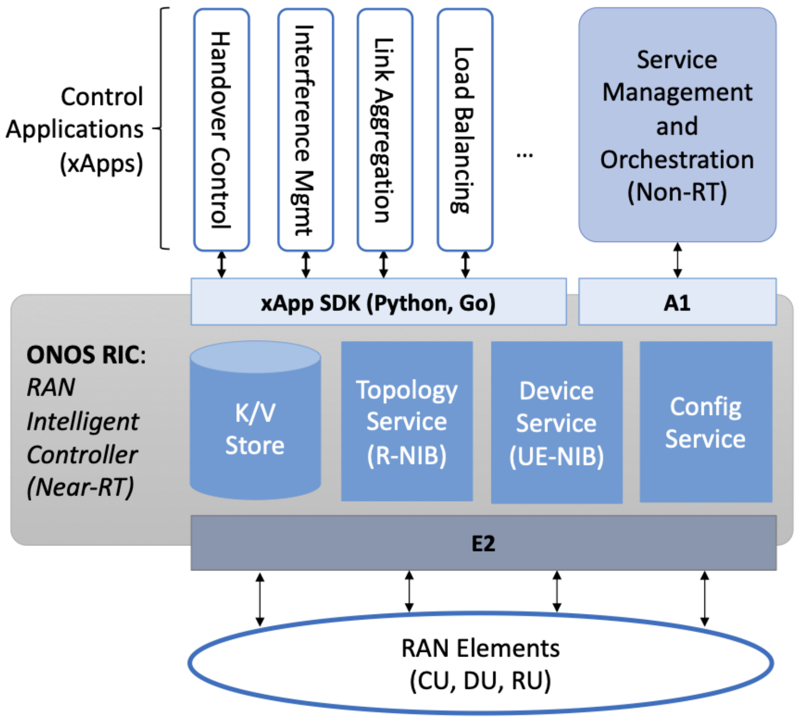

Figure 4.7.: O-RAN compliant RAN Intelligent Controller (RIC) built by adapting and extending ONOS.

Specifically, the ONOS-based RIC includes a Topology Service to keep track of the fixed RAN infrastructure, a Device Service to track the UEs, and a Configuration Service to manage RAN-wide configuration state. All three of these services are implemented as Kubernetes-based microservices, and take advantage of a scalable key-value store.

[2] Technically, the O-RAN definition of the RIC refers to the combination of xApps and the underlying platform (in our case ONOS), but we emphasize the distinction between the two, in keeping with the SDN model of distinguishing between the Network OS and the suite of Control Apps that run on it.

Of the three interfaces called out in Figure 4.7, the **A1** and **E2** interfaces are based on pre-existing 3GPP standards. The third, **xApp SDK**, is specific to the ONOS-based implementation. The O-RAN Alliance is using it to drive towards a unified API (and corresponding SDK) for building RIC-agnostic xApps.

The A1 interface provides a means for the mobile operator's management plane—typically called the *OSS/BSS (Operations Support System / Business Support System)* in the Telco world—to configure the RAN. We briefly introduced the OSS/BSS in Section 2.5, but all you need to know about it for our purposes is that such a component sits at the top of all Telco software stacks. It is the source of all configuration settings and business logic needed to operate a network. You can think of A1 as the RAN's counterpart to gNMI/gNOI (gRPC Network Management Interface/gRPC Network Operations Interface), a pair of configuration APIs commonly used to configure commodity cloud hardware.

The Near-RT RIC uses the E2 interface to control the underlying RAN elements, including the CU, DUs, and RUs. A requirement of the E2 interface is that it be able to connect the Near-RT RIC to different types of RAN elements from different vendors. This range is reflected in the API, which revolves around a *Service Model* abstraction. The idea is that each RAN element advertises a Service Model, which effectively defines the set of RAN Functions the element is able to support. The RIC then issues a combination of the following four operations against this Service Model.

- **Report:** RIC asks the element to report a function-specific value setting.
- **Insert:** RIC instructs the element to activate a user plane function.
- **Control:** RIC instructs the element to activate a control plane function.
- **Policy:** RIC sets a policy parameter on one of the activated functions.

Of course, it is the RAN element, through its published Service Model, that defines the relevant set of functions that can be activated, the variables that can be reported, and policies that can be set. The O-RAN community is working on three vendor-agnostic Service Models. The first, called *Key Performance Measurement* (abbreviated *E2SM-KPM*), specifies the metrics that can be retrieved from RAN elements. The second, called *RAN Control* (abbreviated *E2SM-RC*), specifies parameters that can be set to control RAN elements. The third, called *Cell Configuration and Control* (abbreviated *E2SM-CCC*), exposes configuration parameters at the cell level.

In simple terms, E2SM-KPM defines what values can be *read* and E2SM-RC and E2SM-CCC defines what values can be *written*. Because the available values can be highly variable across all possible devices, we can expect different vendors will support only a subset of the entire collection. This will limit the "universality" the O-RAN was hoping to achieve in an effort to break vendor lock-in, but that outcome is familiar to network operators who have been dealing with divergent *Management Information Bases (MIBs)* since the earliest days of the Internet.

Finally, the xApp SDK, which is specific to the ONOS-based implementation, is currently little more than a "pass through" of the E2 interface. This implies the xApps are expected to be aware of the available Service Models. One of the challenges the SDK has to deal with is how data passed to/from the RAN elements is encoded. For historical reasons, the E2 interface uses ASN.1 formatting, whereas the ONOS-RIC internally uses gRPC and Protocol Buffers to communicate between the set of microservices. The south-bound E2 interface in Figure 4.7 translates between the two formats. The SDK currently makes the gRPC-based API available to xApps.

Further Reading

To learn more about the details of ONOS and its interfaces, we recommend the chapter in our SDN book that covers it in depth. Software-Defined Networks: A Systems Approach. Chapter 6: Network OS.

4.5 4.5 Control Loops

We conclude this description of RAN internals by revisiting the sequence of steps involved in disaggregation, which, as the previous three sections reveal, is being pursued in multiple tiers. In doing so, we tie up several loose ends, and focus attention on the resulting three control loops.

In the first tier of disaggregation, 3GPP defines multiple options for how the RAN can be split and distributed, with the pipeline shown in Figure 4.1 disaggregated into the independently operating CU, DU, and RU components shown in Figure 4.8. The O-RAN Alliance has selected specific disaggregation options from 3GPP and is developing open interfaces between these components.

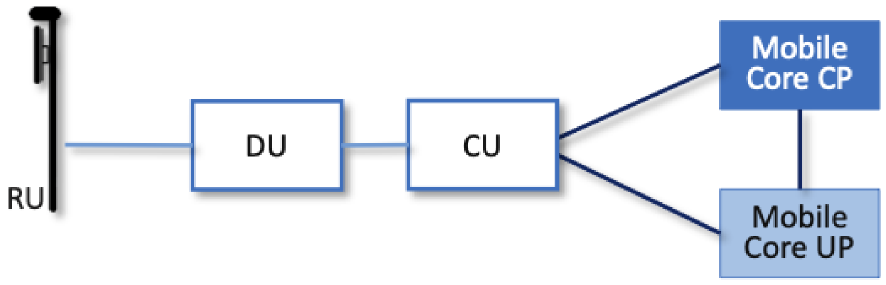

Figure 4.8.: First tier of RAN disaggregation: Split RAN.

The second tier of disaggregation focuses on the control/user plane separation (CUPS) of the CU, resulting in the CU-UP and CU-CP shown in Figure 4.9. The control plane in question is the 3GPP control plane, where the CU-UP realizes a pipeline for user traffic and the CU-CP focuses on control message signaling between Mobile Core and the disaggregated RAN components (as well as to the UE).

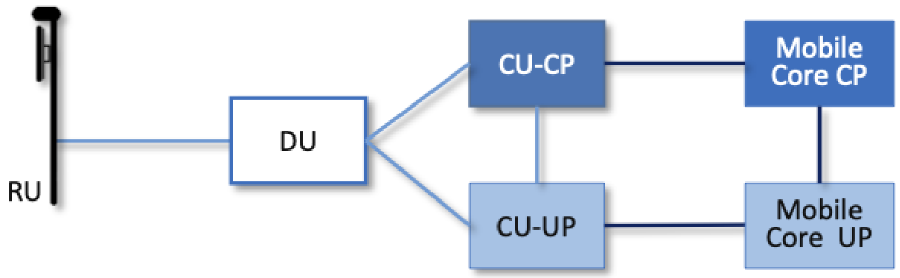

Figure 4.9.: Second tier of RAN disaggregation: CUPS.

The third tier follows the SDN paradigm by separating most of RAN control (RRC functions) from the disaggregated RAN components, and logically centralizing them as applications running on an SDN Controller, which corresponds to the Near-RT RIC shown previously in Figures 4.5 and 4.6. This SDN-based disaggregation is repeated in Figure 4.10, which also shows the O-RAN-prescribed interfaces A1 and E2 introduced in the previous section. (Note that all the edges in Figures 4.8 and 4.9 correspond to 3GPP-defined interfaces, as identified in Section 4.2, but their details are outside the scope of this discussion.)

Taken together, the A1 and E2 interfaces complete two of the three major control loops of the RAN: the outer (non-real-time) loop has the Non-RT RIC as its control point and the middle (near-real-time) loop has the Near-RT RIC as its control point. The third (innermost) control loop—shown in Figure 4.10 running inside the DU—includes the real-time Scheduler embedded in the MAC stage of the RAN pipeline. The two outer control loops have rough time bounds of >>1sec and >10ms, respectively. As we saw in Chapter 2, the real-time control loop is assumed to be <1ms.

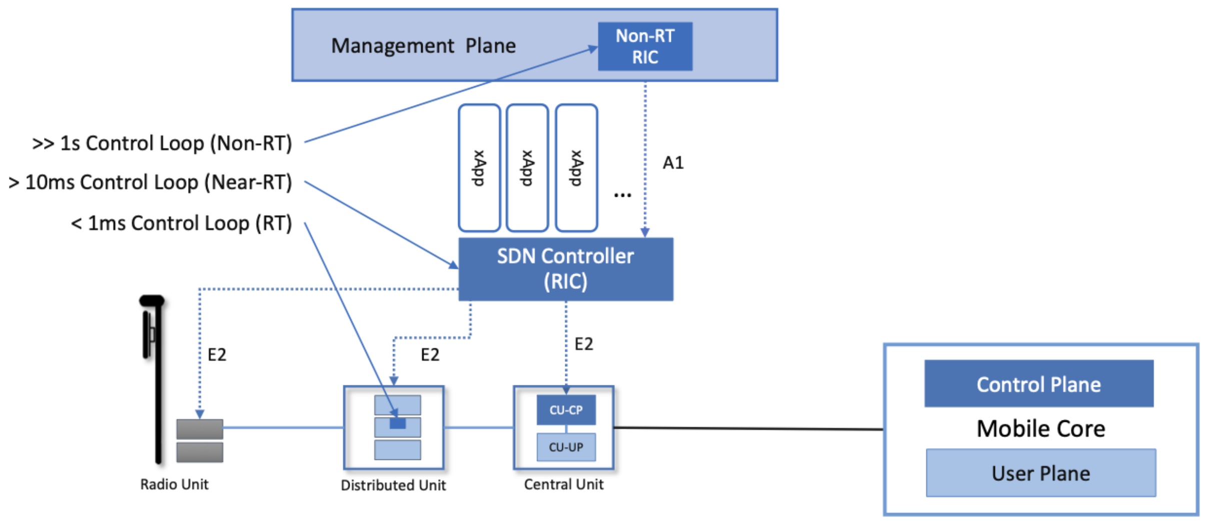

Figure 4.10.: Third tier of RAN disaggregation: SDN.

This raises the question of how specific functionality is distributed between the Non-RT RIC, Near-RT RIC, and DU. Starting with the second pair (i.e., the two inner loops), it is the case that not all RRC functions can be centralized; some need to be implemented in the DU. The SDN-based disaggregation then focuses on those that can be centralized, with the Near-RT RIC supporting the RRC applications and the SON applications mentioned in Section 4.3.

Turning to the outer two control loops, the Near RT-RIC opens the possibility of introducing policy-based RAN control, whereby interrupts (exceptions) to operator-defined policies would signal the need for the outer loop to become involved. For example, one can imagine developing learning-based controls, where the inference engines for these controls would run as applications on the Near RT-RIC, and their non-real-time learning counterparts would run elsewhere. The Non-RT RIC would then interact with the Near-RT RIC to deliver relevant operator policies from the Management Plane to the Near RT-RIC over the A1 interface.

CHAPTER

FIVE

CHAPTER 5: MOBILE CORE

The Mobile Core provides IP connectivity to the RAN. It authenticates UEs as they connect, tracks them as they move from one base station to another, ensures that this connectivity fulfills the promised QoS requirements, and meters usage for billing.

Historically, all of these functions were provided by one or more proprietary network appliances. But like the rest of the 5G mobile network, these appliances are being disaggregated and implemented as a set of cloud services, with the goal of improving feature velocity for new classes of applications. It is also the case that as the range of use cases grows more diverse, a one-size-fits-all approach will become problematic. The expectation is that it should be possible to customize and specialize the Mobile Core on a per-application basis.

This chapter introduces the functional elements of the Mobile Core, and describes different strategies for implementing that functionality.

5.1 5.1 Identity Management

There are two equally valid views of the Mobile Core. The Internet-centric view is that each local instantiation of the Mobile Core (e.g., serving a metro area) acts as a router that connects a physical RAN (one of many possible access network technologies, not unlike WiFi) to the global Internet. In this view, IP addresses serve as the unique global identifier that makes it possible for any RAN-connected device to communicate with any Internet-addressable device or service. The 3GPP-centric view is that a distributed set of Mobile Cores (interconnected by one or more backbone technologies, of which the Internet is just one example) cooperate to turn a set of physical RANs into one logically global RAN. In this perspective, the IMSI burned into a device's SIM card serves as the global identifier that makes it possible for any two mobile devices to communicate with each other.

Both of these perspectives are correct, but since broadband communication using Internet protocols to access cloud services is the dominant use case, this section takes an Internet-centric perspective of the Mobile Core. But before getting to that, we first need to understand several things about the 3GPP-centric perspective.

For starters, we need to be aware of the distinction between "identity" and "identifier". The first term is commonly used when talking about principals or users, and the second term is used when talking about abstract objects or physical devices. Unfortunately, the two terms are conflated in the 3GPP architecture: The acronym IMSI explicitly includes the word "Identity", where the "S" in both IMSI and SIM stands for subscriber (a kind of principal), yet the IMSI is also used as a global identifier for a UE connected to the mobile network. This conflation breaks down when there could be tens or hundreds of IoT devices for every customer, with no obvious association among them. Accounting for this problem is an "architecture alignment" fix we discuss in the next chapter when we describe how to provide Private 5G Connectivity as a managed cloud service.

If we take the view that an IMSI is primarily a global identifier for UEs, then we can think of it as the mobile network's equivalent of a 48-bit 802.3 or 802.11 MAC address. This includes how addresses are assigned to ensure uniqueness: (MCC, MNC) pairs are assigned by a global authority to every MNO, each of which then decides how to uniquely assign the rest of the IMSI identifier space to devices. This approach is similar to how network vendors are assigned a unique prefix for all the MAC addresses they configure into the NIC cards and WiFi chips they ship, but with one big difference: It is the MNO, rather than the vendor, that is responsible for assigning IMSIs to SIM cards. This makes the IMSI allocation problem closer to how the Internet assigns IP addresses to end hosts, but unlike DHCP, the IMSI-to-device binding is static.

This is important because, unlike 802.11 addresses, IMSIs are also intended to support global routing. (Here, we are using a liberal notion of routing—to locate an object—and focusing on the original 3GPP-perspective of the global RAN in which the Internet is just a possible packet network that interconnects Mobile Cores.) A hierarchically distributed database maps IMSIs onto the collection of information needed to forward data to the corresponding device. This includes a combination of relatively *static* information about the level of service the device expects to receive (including the corresponding phone number and subscriber profile/account information), and more *dynamic* information about the current location of the device (including which Mobile Core, and which base station served by that Core, currently connects the device to the global RAN).

This mapping service has a name, or rather, several names that keep changing from generation to generation. In 2G and 3G it was called HLR (Home Location Registry). In 4G the HLR maintains only static information and a separate HSS (Home Subscriber Server) maintains the more dynamic information. In 5G the HLR is renamed the UDR (Unified Data Registry) and the HSS is renamed UDM (Unified Data Management). We will see the UDM in Section 5.2 because of the role it plays *within* a single instance of the Mobile Core.

There are, of course, many more details to the process—including how to find a device that has roamed to another MNO's network—but conceptually the process is straightforward. (As a thought experiment, imagine how you would build a "logically global WiFi" using just 802.11 addresses, rather than depending on the additional layer of addressing provided by IP.) The important takeaway is that IMSIs are used to locate the Mobile Core instance that is then responsible for authenticating the device, tracking the device as it moves from base station to base station within that Core's geographic region, and forwarding packets to/from the device.

Two additional observations about the relationship between IMSIs and IP addresses are worth highlighting. First, the odds of someone trying to "call" or "text" an IoT device, drone, camera, or robot are virtually zero. It is the IP address assigned to each device (by the local Mobile Core) that is used to locate (route packets to) the device. In this context, the IMSI plays exactly the same role in a physical RAN as an 802.11 address plays in a LAN, and the Mobile Core behaves just like any access router.

Second, whether a device connects to a RAN or some other access network, it is automatically assigned a new IP address any time it moves from one coverage domain to another. Even for voice calls in the RAN case, ongoing calls are dropped whenever a device moves between instantiations of the Mobile Core (i.e., uninterrupted mobility is supported only within the region served by a given Core). This is typically not a problem when the RAN is being used to deliver broadband connectivity because Internet devices are almost always clients *requesting* a cloud service; they just start issuing requests with their new (dynamically assigned) IP address.

5.2 5.2 Functional Components

The 5G Mobile Core, which 3GPP calls the *5GC*, adopts a microservice-like architecture officially known as the 3GPP *Service Based Architecture*. We say "microservice-like" because while the 3GPP specification spells out this level of disaggregation, it is really just describing a set of functional blocks and not prescribing an implementation. In practice, a set of functional blocks is very different from the collection of engineering decisions that go into designing a microservice-based system. That said, viewing the collection of components shown in Figure 5.1 as a set of microservices is a reasonable working model (for now).

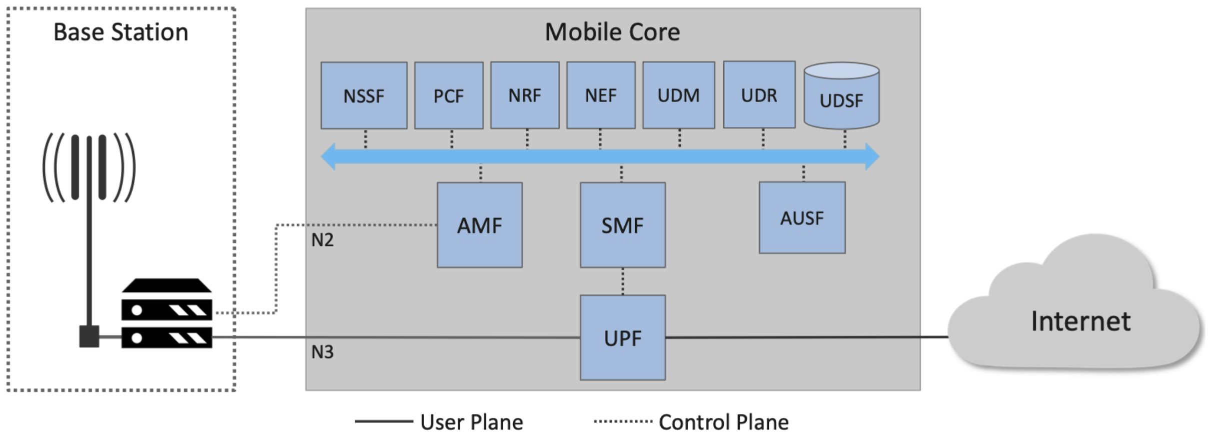

Figure 5.1.: 5G Mobile Core (5GC), represented as a collection of microservices, where 3GPP defines the interfaces connecting the Mobile Core CP and UP to the RAN (denoted N2 and N3, respectively).

Starting with the User Plane (UP), the *UPF (User Plane Function)* forwards traffic between the RAN and the Internet. In addition to IP packet forwarding, the UPF is responsible for policy enforcement, lawful intercept, traffic usage measurement, and QoS policing. These are all common functions in access routers, even if they go beyond what you usually find in enterprise or backbone routers. The other detail of note is that, because the RAN is an overlay network, the UPF is responsible for tunneling (i.e., encapsulating and decapsulating) packets as they are transmitted to and from base stations over the N3 interface (as depicted in Figure 2.8 of Section 2.3).

The rest of the functional elements in Figure 5.1 implement the Control Plane (CP). Of these, two represent the majority of the functionality that's unique to the Mobile Core CP (as sketched in Figure 2.11 of Section 2.4):

- *AMF (Access and Mobility Management Function):* Responsible for connection and reachability management, mobility management, access authorization, and location services.
- *SMF (Session Management Function):* Manages each UE session, including IP address allocation, selection of associated UP function, control aspects of QoS, and control aspects of UP routing.

In other words, the AMF authorizes access when a UE first connects to one of the local base stations, and then tracks (but does not control) which base station currently serves each UE. The SMF then allocates an IP address to each AMF-authorized UE, and directly interacts with the UPF to maintain per-device session state.

Of particular note, the per-UE session state controlled by the SMF (and implemented by the UPF) includes a packet buffer in which packets destine to an idle UE are queued during the time the UE transitions to active state. This feature was originally designed to avoid data loss during a voice call, but its value is less obvious when the data is an IP packet since end-to-end protocols like TCP are prepared to retransmit lost packets. On the other hand, if idle-to-active transitions are too frequent, they can be problematic for TCP.

Before continuing with our inventory of control-related elements in Figure 5.1, it is important to note we show only a fraction of the full set that 3GPP defines. The full set includes a wide range of possible features, many of which are either speculative (i.e., identify potential functionality) or overly prescriptive (i.e., identify well-known cloud native microservices). We limit our discussion to functional elements that provide value in the private 5G deployments that we focus on. Of these, several provide functionality similar to what one might find in any microservice-based application:

- *AUSF (Authentication Server Function):* Authenticates UEs.
- *UDM (Unified Data Management):* Manages user identity, including the generation of authentication credentials and access authorization.
- *UDR (Unified Data Repository):* Manages user static subscriber-related information.
- *UDSF (Unstructured Data Storage Function):* Used to store unstructured data, and so is similar to a *key-value store*.
- *NEF (Network Exposure Function):* Exposes select capabilities to third-party services, and so is similar to an *API Server*.
- *NRF (Network Repository Function):* Used to discover available services (network functions), and so is similar to a *Discovery Service*.

The above list includes 3GPP-specified control functions that are, in some cases, similar to well-known microservices. In such cases, substituting an existing cloud native component is a viable implementation option. For example, MongoDB can be used to implement a UDSF. In other cases, however, such a one-for-one swap is not possible due to assumptions 3GPP makes. For example, AUSF, UDM, UDR, and AMF collectively implement a *Authentication and Authorization Service*, but an option like OAuth2 could not be used in their place because (a) UDM and UDR are assumed to be part of the global identity mapping service discussed in Section 5.1, and (b) 3GPP specifies the interface by which the various components request service from each other (e.g., AMF connects to the RAN via the N2 interface depicted in Figure 5.1). We will see how to cope with such issues in Section 5.3, where we talk about implementation issues in more detail.

Finally, Figure 5.1 shows two other functional elements that export a northbound interface to the management plane (not shown):

- *PCF (Policy Control Function):* Manages the policy rules for the rest of the Mobile Core CP.
- *NSSF (Network Slice Selection Function):* Manages how network slices are selected to serve a given UE.

Keep in mind that even though 3GPP does not directly prescribe a microservice implementation, the overall design clearly points to a cloud native solution as the desired end-state for the Mobile Core. Of particular note, introducing a distinct storage service means that all the other services can be stateless, and hence, more readily scalable.

5.3 5.3 Control Plane

This section describes two different strategies for implementing the Mobile Core CP. Both correspond to open source projects that are available for download and experimentation.

5.3.1 5.3.1 SD-Core

Our first example, called SD-Core, is a nearly one-for-one translation of the functional blocks shown in Figure 5.1 into a cloud native implementation. A high-level schematic is shown in Figure 5.2, where each element corresponds to a scalable set of Kubernetes-hosted containers. We include this schematic even though it looks quite similar to Figure 5.1 because it highlights four implementation details.

Further Reading

SD-Core.

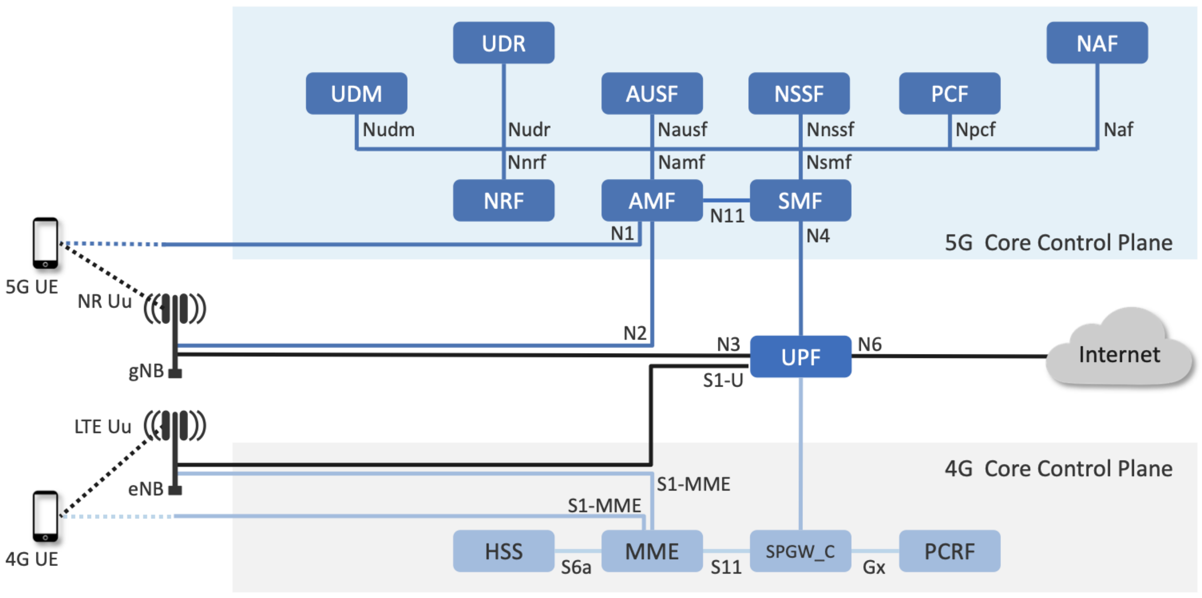

Figure 5.2.: SD-Core implementation of the Mobile Core Control Plane, including support for Standalone (SA) deployment of both 4G and 5G.

First, SD-Core supports both the 5G and 4G versions of the Mobile Core,[1] which share a common User Plane (UPF). We have not discussed details of the 4G Core, but observe that it is less disaggregated. In particular, the components in the 5G Core are specified so that they can be stateless, simplifying the task of horizontally scaling them out as load dictates. (The rough correspondence between 4G and 5G is: MME-to-AMF, SPGW_C-to-SMF, HSS-to-UDM, and PCRF-to-PCF.) Although not shown in the schematic, there is also a scalable key-value store microservice based on MongoDB. It is used to make Core-related state persistent for the Control Planes; for example, UDM/UDR (5G) and HSS (4G) write subscriber state to MongoDB.

Second, Figure 5.2 illustrates 3GPP's *Standalone (SA)* deployment option, in which 4G and 5G networks co-exist and run independently. They share a UPF implementation, but UPF instances are instantiated separately for each RAN/Core pair, with support for both the 4G and 5G interfaces, denoted *S1-U* and *N3*,

[1] SD-Core's 4G Core is a fork of the OMEC project and its 5G Core is a fork of the Free5GC project.

respectively. Although not obvious from the SA example, 3GPP defines an alternative transition plan, called *NSA (Non-Standalone)*, in which separate 4G and 5G RANs were paired with either a 4G Core or a 5G Core. The details of how that works are not relevant to this discussion, except to make the point that production networks almost never get to enjoy a "flag day" on which a new version is universally substituted for an old version. A migration plan has to be part of the design. More information on this topic can be found in a GSMA Report.

Further Reading

Road to 5G: Introduction and Migration. GSMA Report, April 2018.

Third, Figure 5.2 shows many of the 3GPP-defined inter-component interfaces. These include an over-the-air interface between base stations and UEs (*NR Uu*), control interfaces between the Core and both UEs and base stations (*N1* and *N2*, respectively), a user plane interface between the Core and base stations (*N3*), and a data plane interface between the Core and the backbone network (*N6*).

The schematic also shows interfaces between the individual microservices that make up the Core's Control Plane; for example, *Nudm* is the interface to the UDM microservice. These latter interfaces are RESTful, meaning clients access each microservice by issuing GET, PUT, POST, PATCH, and DELETE operations over HTTP, where a service-specific schema defines the available resources that can be accessed. Note that some of these interfaces are necessary for interoperability (e.g., *N1* and *N Uu* make it possible to connect your phone to any MNO's network), but others could be seen as internal implementation details. We'll see how Magma takes advantage of this distinction in the next section.

Fourth, by adopting a cloud native design, SD-Core benefits from being able to horizontally scale individual microservices. But realizing this benefit isn't always straightforward. In particular, because the AMF is connected to the RAN by SCTP (corresponding to the *N1* and *N2* interfaces shown in Figure 5.2), it is necessary to put an *SCTP load balancer* in front of the AMF. This load balancer terminates the SCTP connections, and distributes requests across a set of AMF containers. These AMF instances then depend on a scalable backend store (specifically MongoDB) to read and write shared state.

5.3.2 5.3.2 Magma

Magma is an open source Mobile Core implementation that takes a different and slightly non-standard approach. Magma is similar to SD-Core in that it is implemented as a set of microservices, but it differs in that it is designed to be particularly suitable for remote and rural environments with poor backhaul connectivity. This emphasis, in turn, leads Magma to (1) adopt an SDN-inspired approach to how it separates functionality into centralized and distributed components, and (2) factor the distributed functionality into microservices without strict adherence to all the standard 3GPP interface specifications. This refactoring is also a consequence of Magma being designed to unify 4G, 5G, and WiFi under a single architecture.

One of the first things to note about Magma is that it takes a different view of "backhaul" from the approaches we have seen to date. Whereas the backhaul networks shown previously connect the eNBs/gNBs and radio towers back to the mobile core (Figure 2.1), Magma actually puts much of the mobile core functionality right next to the radio as seen in Figure 5.4. It is able to do this because of the way it splits the core into centralized and distributed parts. So Magma views "backhaul" as the link that connects a remote deployment to the rest of the Internet (including the central components), contrasting with conventional 3GPP usage. As explored further below, this can overcome many of the challenges that unreliable backhaul links introduce in conventional approaches.

Figure 5.3 shows the overall Magma architecture. The central part of Magma is the single box in the

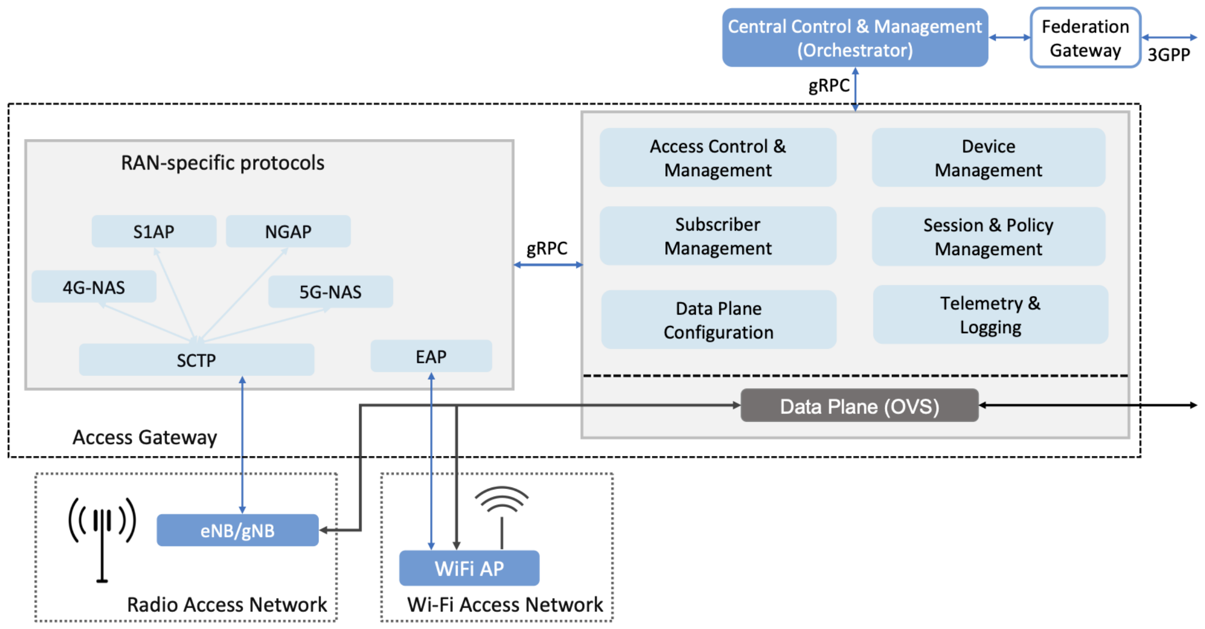

Figure 5.3.: Overall architecture of the Magma Mobile Core, including support for 4G and 5G, and Wi-Fi. There is one central Orchestrator and typically many Access Gateways (AGWs).

figure marked *Central Control & Management (Orchestrator)*. This is roughly analogous to the central controller found in typical SDN systems, and provides a northbound API by which an operator or other software systems (such as a traditional OSS/BSS) can interact with the Magma core. The orchestrator communicates over backhaul links with Access Gateways (AGWs), which are the distributed components of Magma. A single AGW typically handles a small number of eNBs/gNBs. As an example, see Figure 5.4 which includes a single eNB and AGW located on a radio tower. In this example, a point-to-point wireless link is used for backhaul.

The AGW is designed to have a small footprint, so that small deployments do not require a datacenter's worth of equipment. Each AGW also contains both data plane and control plane elements. This is a little different from the classic approach to SDN systems in which only the data plane is distributed. Magma can be described as a hierarchical SDN approach, as the control plane itself is divided into a centralized part (running in the Orchestrator) and a distributed part (running in the AGW). Figure 5.3 shows the distributed control plane components and data plane in detail. We postpone a general discussion of orchestration until Chapter 6.

Magma differs from the standard 3GPP approach in that it terminates 3GPP protocols logically close to the edge, which in this context corresponds to two interface points: (1) the radio interface connecting Magma to an eNB or gNB (implemented by set of modules on the left side of the AGW in the figure) or the federation interface connecting Magma to another mobile network (implemented by the *Federation Gateway* module in the figure). Everything "between" those two external interfaces is free to deviate from the 3GPP specification, which has a broad impact as discussed below.

One consequence of this approach is that Magma can interoperate with other implementations *only* at the edges. Thus, it is possible to connect a Magma mobile core to any standards-compliant 4G or 5G base station and expect it to work, and similarly, it is possible to federate a Magma core with an existing MNO's 4G or 5G network. However, since Magma does not implement all the 3GPP interfaces that are internal to a mobile packet core, it is not possible to arbitrarily mix and match components within the core. Whereas (in principle) a traditional 3GPP implementation would permit an AMF from one vendor to interoperate with the SMF of another vendor, it is not possible to connect parts of a mobile core from another vendor (or another open source project) with parts of Magma, aside from via the two interfaces

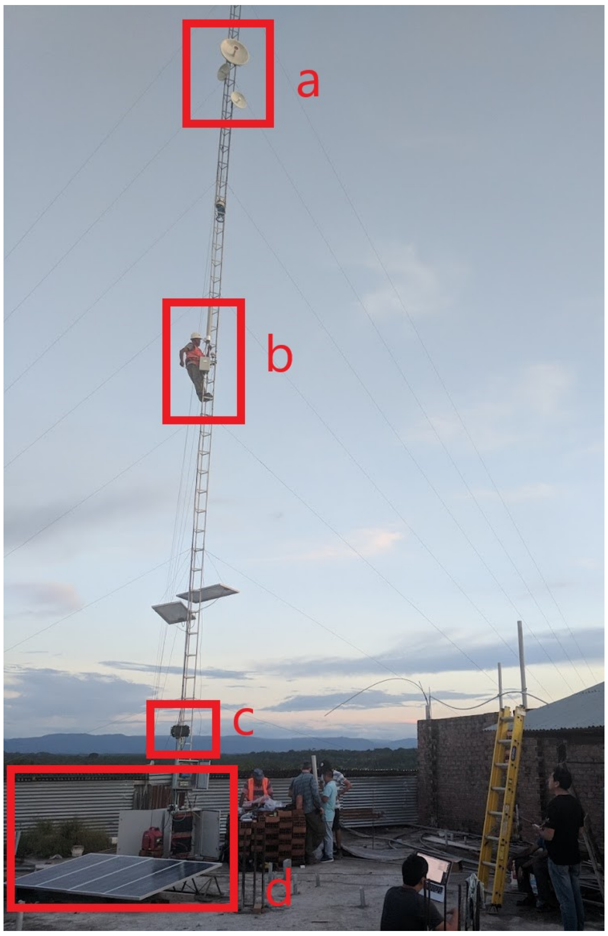

Figure 5.4.: A sample Magma deployment in rural Peru, showing (a) point-to-point wireless backhaul, (b) LTE radio and antenna, (c) ruggedized embedded PC serving as AGW, and (d) solar power and battery backup for site.

just described.

Being free to deviate from the 3GPP spec means Magma can take a unifying approach across multiple wireless technologies, including 4G, 5G and WiFi. There is a set of functions that the core must implement for any radio technology (e.g., finding the appropriate policy for a given subscriber by querying a database); Magma provides them in an access-technology-independent way. These functions form the heart of an Access Gateway (AGW), as illustrated on the right side of Figure 5.3. On the other hand, control protocols that are specific to a given radio technology are terminated in technology-specific modules close to the radio. For example, *SCTP* shown on the left side of the figure is the RAN tunneling protocol introduced in Section 2.3. These technology-specific modules then communicate with the generic functions (e.g., subscriber management, access control and management) on the right using gRPC messages that are technology-agnostic.

Magma's design is particularly well suited for environments where backhaul links are unreliable, for example, when a satellite is used. This is because the 3GPP protocols that traditionally have to traverse the backhaul from core to eNB/gNB are quite sensitive to loss and latency. Loss or latency can cause connections to be dropped, which in turn forces UEs to repeat the process of attaching to the core. In practice, not all UEs handle this elegantly, sometimes ending up in a "stuck" state.

Magma addresses the challenge of unreliable backhaul in several ways. First, Magma frequently avoids sending messages over the backhaul entirely by running more functionality in the AGW, which is located close to the radio as seen above. Functions that would be centralized in a conventional 3GPP implementation are distributed out to the access gateways in Magma. Thus, for example, the operations required to authenticate and attach a UE to the core can typically be completed using information cached locally in the AGW, without any traffic crossing the backhaul. Secondly, when Magma does need to pass information over a backhaul link (e.g., to obtain configuration state from the orchestrator), it does so using gRPC, which is designed to operate reliably in the face of unreliable or high-latency links.

Note that while Magma has distributed much of the control plane out to the AGWs, it still supports centralized management via the Orchestrator. For example, adding a new subscriber to the network is done centrally, and the relevant AGW then obtains the necessary state to authenticate that subscriber when their UE tries to attach to the network.

Finally, Magma adopts a *desired state* model for managing runtime and configuration state. By this we mean that it communicates a state change (e.g., the addition of a new session in the user plane) by specifying the desired end state via an API call. This is in contrast with the *incremental update* model that is common in the 3GPP specification. When the desired end state is communicated, the loss of a message or failure of a component has less serious consequences. This makes reasoning about changes across elements of the system more robust in the case of partial failures, which are common in challenged environments like the ones Magma is designed to serve.

Consider an example where we are establishing user plane state for a set of active sessions. Initially, there are two active sessions, X and Y. Then a third UE becomes active and a session Z needs to be established. In the incremental update model, the control plane would instruct the user plane to "add session Z". The desired state model, by contrast, communicates the entire new state: "the set of sessions is now X, Y, Z". The incremental model is brittle in the face of failures. If a message is lost, or a component is temporarily unable to receive updates, the receiver falls out of sync with the sender. So it is possible that the control plane believes that sessions X, Y and Z have been established, while the user plane has state for only X and Y. By sending the entire desired end state, Magma ensures that the receiver comes back into sync with the sender once it is able to receive messages again.

As described, this approach might appear inefficient because it implies sending complete state information rather than incremental updates. However, at the scale of an AGW, which handles on the order of hundreds to a few thousands of subscribers, it is possible to encode the state efficiently enough to overcome this drawback. With the benefit of experience, mechanisms have been added to Magma to avoid overloading the orchestrator, which has state related to all subscribers in the network.

The desired state approach is hardly novel but differs from typical 3GPP systems. It allows Magma to tolerate occasional communication failures or component outages due to software restarts, hardware failures, and so on. Limiting the scope of 3GPP protocols to the very edge of the network is what enables Magma to rethink the state synchronization model. The team that worked on Magma describes their approach in more detail in an NSDI paper.

Further Reading

S. Hasan, *et al.* Building Flexible, Low-Cost Wireless Access Networks With Magma. NSDI, April 2023.

Finally, while we have focused on its Control Plane, Magma also includes a User Plane component. The implementation is fairly simple, and is based on Open vSwitch (OVS). Having a programmable user plane is important, as it needs to support a range of access technologies, and at the same time, OVS meets the performance needs of AGWs. However, this choice of user plane is not fundamental to Magma, and other implementations have been considered. We take a closer look at the User Plane in the next section.

5.4 5.4 User Plane

The User Plane of the Mobile Core—corresponding to the UPF component in Figure 5.1—connects the RAN to the Internet. Much like the data plane for any router, the UPF forwards IP packets, but because UEs often sleep to save power and may be in the process of being handed off from one base station to another, it sometimes has to buffer packets for an indeterminate amount of time. Also like other routers, a straightforward way to understand the UPF is to think of it as implementing a collection of Match/Action rules, where the UPF first classifies each packet against a set of matching rules, and then executes the associated action.

Using 3GPP terminology, packet classification is defined by a set of *Packet Detection Rules (PDRs)*, where a given PDR might simply match the device's IP address, but may also take the domain name of the far end-point into consideration. Each attached UE has at least two PDRs, one for uplink traffic and one for downlink traffic, plus possibly additional PDRs to support multiple traffic classes (e.g., for different QoS levels, pricing plans, and so on.). The Control Plane creates, updates, and removes PDRs as UEs attach, move, and detach.

Each PDR then identifies one or more actions to execute, which in 3GPP terminology are also called "rules", of which there are four types:

- **Forwarding Action Rules (FARs):** Instructs the UPF to forward downlink packets to a particular base station and uplink traffic to a next-hop router. Each FAR specifies a set of parameters needed to forward the packet (e.g., how to tunnel downlink packets to the appropriate base station), plus one of the following processing flags: a *forward* flag indicates the packet should be forwarded up to the Internet; a *tunnel* flag indicates the packet should be tunneled down to a base station; a *buffer* flag indicates the packet should be buffered until the UE becomes active; and a *notify* flag indicates that the CP should be notified to awaken an idle UE. FARs are created and removed when a device attaches or detaches, respectively, and the downlink FAR changes the processing flag when the device moves, goes idle, or awakes.
- **Buffering Action Rules (BARs):** Instructs the UPF to buffer downlink traffic for idle UEs, while also sending a *Downlink Data Notification* to the Control Plane. This notification, in turn, causes the CP to instruct the base station to awaken the UE. Once the UE becomes active, the UPF releases the buffered traffic and resumes normal forwarding. The buffering and notification functions are activated by modifying a FAR to include *buffer* and *notify* flags, as just described. An additional set of parameters are used to configure the buffer, for example setting its maximum size (number of bytes) and duration (amount of time). Optionally, the CP can itself buffer packets by creating a PDR that directs the UPF to forward data packets to the control plane.
- **Usage Reporting Rules (URRs):** Instructs the UPF to periodically send usage reports for each UE to the CP. These reports include counts of the packets sent/received for uplink/downlink traffic for each UE and traffic class. These reports are used to both limit and bill subscribers. The CP creates and removes URRs when the device attaches and detaches, respectively, and each URR specifies whether usage reports should be sent periodically or when a quota is exceeded. A UE typically has two URRs (for uplink/downlink usage), but if a subscriber's plan includes special treatment for certain types of traffic, an additional URR is created for each such traffic class.
- **Quality Enforcement Rules (QERs):** Instructs the UPF to guarantee a minimum amount of bandwidth and to enforce a bandwidth cap. These parameters are specified on a per-UE / per-direction / per-class basis. The CP creates and removes QERs when a device attaches and detaches, respectively, and modifies them according to operator-defined events, such as when the network becomes more or less congested, the UE exceeds a quota, or the network policy changes (e.g., the user signs up for a new pricing plan). The UPF then performs traffic policing to enforce the bandwidth cap, along with packet scheduling to ensure a minimum bandwidth in conjunction with admission

control in the control plane.

The rest of this section describes two complementary strategies for implementing a UPF, one server-based and one switch-based.

5.4.1 5.4.1 Microservice Implementation

A seemingly straightforward approach to supporting the set of Match/Action rules just described is to implement the UPF in software on a commodity server. Like any software-based router, the process would read a packet from an input port, classify the packet by matching it against a table of configured PDRs, execute the associated action(s), and then write the packet to an output port. Such a process could then be packaged as a Docker container, with one or more instances spun up on a Kubernetes cluster as workload dictates. This is mostly consistent with a microservice-based approach, with one important catch: the actions required to process each packet are stateful.

What we mean by this is that the UPF has two pieces of state that needs to be maintained on a per-UE / per-direction / per-class basis: (1) a finite state machine that transitions between *forward*, *tunnel*, *buffer*, and *notify*; and (2) a corresponding packet buffer when in *buffer* state. This means that as the UPF scales up to handle more and more traffic—by adding a second, third, and fourth instance—packets still need to be directed to the original instance that knows the state for that particular flow. This breaks a fundamental assumption of a truly horizontally scalable service, in which traffic can be randomly directed to any instance in a way that balances the load. It also forces you to do packet classification before selecting which instance is the right one, which can potentially become a performance bottleneck, although it is possible to offload the classification stage to a SmartNIC/IPU.

5.4.2 5.4.2 P4 Implementation

Since the UPF is fundamentally an IP packet forwarding engine, it can also be implemented—at least in part—as a P4 program running on a programmable switch. Robert MacDavid and colleagues describe how that is done in SD-Core, which builds on the base packet forwarding machinery described in our companion SDN book. For the purposes of this section, the focus is on the four main challenges that are unique to implementing the UPF in P4.

Further Reading

R. MacDavid, *et al.* A P4-based 5G User Plane Function. Symposium on SDN Research, September 2021.

Software-Defined Networks: A Systems Approach. November 2021.

First, P4-programmable forwarding pipelines include an explicit "matching" mechanism built on *Ternary Content-Addressable Memory (TCAM)*. This memory supports fast table lookups for patterns that include wildcards, making it ideal for matching IP prefixes. In the case of the UPF, however, the most common PDRs correspond to exact matches of IP addresses (for downlink traffic to each UE) and GTP tunnel identifiers (for uplink traffic from each UE). More complex PDRs might include regular expressions for DNS names or require deep packet inspection.

Because TCAM capacity is limited, and the number of unique PDRs that need to be matched in both directions is potentially in the tens of thousands, it's necessary to use the TCAM judiciously. One implementation strategy is to set up two parallel PDR tables: one using the relatively plentiful switch SRAM for common-case uplink rules that exactly matches on tunnel identifiers (which can be treated as table indices); and one using TCAM for common-case downlink rules that match the IP destination address.

Second, when a packet arrives from the Internet destined for an idle UE, the UPF buffers the packet and sends an alert to the 5G control plane, asking that the UE be awakened. Today's P4-capable switches do not have large buffers or the ability to hold packets indefinitely, but a buffering microservice running on a server can be used to address this limitation. The microservice indefinitely holds any packets that it receives, and later releases them back to the switch when instructed to do so. The following elaborates on how this would work.

When the Mobile Core detects that a UE has gone idle (or is in the middle of a handover), it creates a FAR with the *buffer* flag set, causing the on-switch P4 program to redirect packets to the buffering microservice. Packets are redirected without modifying their IP headers by placing them in a tunnel, using the same tunneling protocol that is used to send data to base stations. This allows the switch to treat the buffering microservice just like another base station.

When the first packet of a flow arrives at the buffering microservice, it sends an alert to the CP, which then (1) wakes up the UE, (2) modifies the corresponding FAR by unsetting the *buffer* flag and setting the *tunnel* flag, and once the UE is active, (3) instructs the buffering microservice to release all packets back to the switch. Packets arriving at the switch from the buffering microservice skip the portion of the UPF module they encountered before buffering, giving the illusion they are being buffered in the middle of the switch. That is, their processing resumes at the tunneling stage, where they are encapsulated and routed to the appropriate base station.

Third, QERs cannot be fully implemented in the switch because P4 does not include support for programming the packet scheduler. However, today's P4 hardware does include fixed-function schedulers with configurable weights and priorities; these parameters are set using a runtime interface unrelated to P4. A viable approach, similar to the one MacDavid, Chen, and Rexford describe in their INFOCOM paper, is to map each QoS class specified in a QER onto one of the available queues, and assign a weight to that queue proportional to the fraction of the available bandwidth the class is to receive. As long as each class/queue is not over subscribed, individual UEs in the class will receive approximately the bit rate they have been promised. As an aside, since 3GPP under-specifies QoS guarantees (leaving the details to the implementation), such an approach is 3GPP-compliant.

Further Reading

R. MacDavid, X. Chen, J. Rexford. Scalable Real-time Bandwidth Fairness in Switches. IEEE INFOCOM, May 2023.

Finally, while the above description implies the Mobile Core's CP talks directly to the P4 program on the switch, the implementation is not that straightforward. From the Core's perspective, the SMF is responsible for sending/receiving control information to/from the UPF, but the P4 program implementing the UPF is controlled through an interface (known as P4Runtime or P4RT) that is auto-generated from the P4 program being controlled. MacDavid's paper describes how this is done in more detail (and presumes a deep understanding of the P4 toolchain), but it can be summarized as follows. It is necessary to first write a "Model UPF" in P4, use that to program to generate the UPF-specific P4RT interface, and then write translators that (1) connect SMF to P4RT, and (2) connect P4RT to the underlying physical switches and servers. A high-level schematic of this software stack is shown in Figure 5.5.

Note that while this summary focuses on how the CP controls the UPF (the downward part of the schematic shown in Figure 5.5), the usage counters needed to generate URRs that flow upward to the CP are easy to support because the counters implemented in the switching hardware are identical to the counters in the Model UPF. When the Mobile Core requests counter values from the Model UPF, the

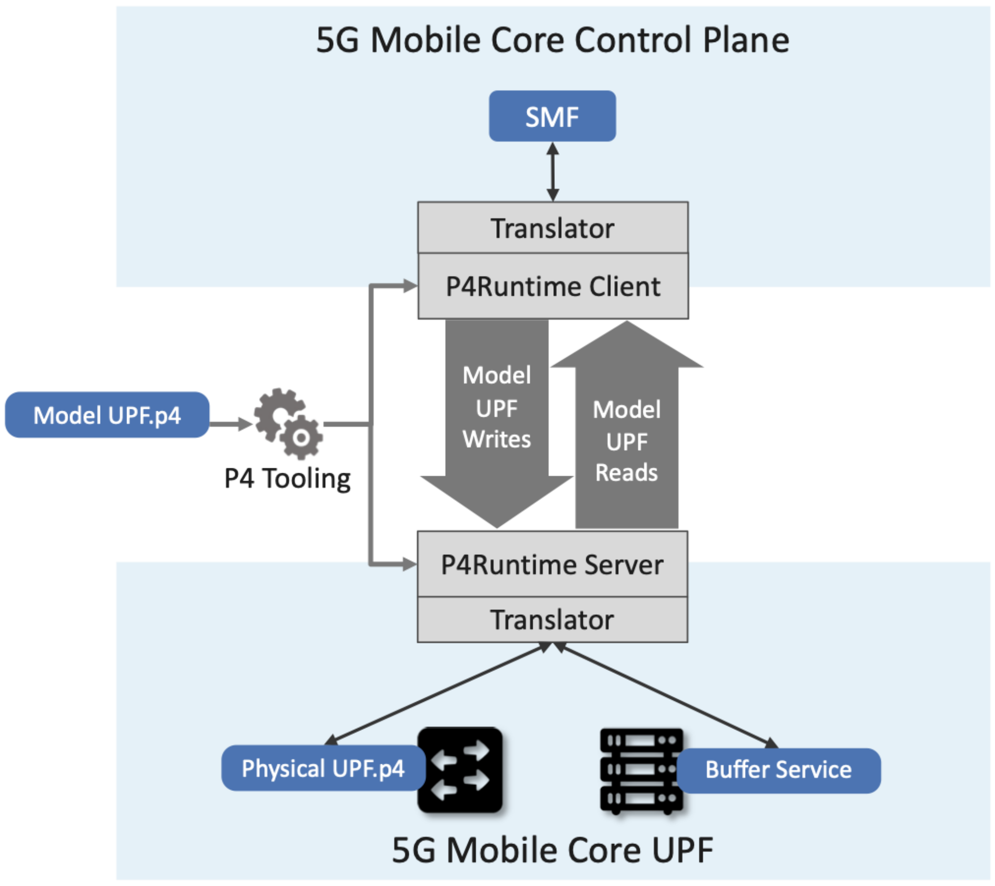

Figure 5.5.: A model P4-based implementation of the UPF is used to generate the interface that is then used by the SMF running in the Mobile Core control plane to control the physical implementation of the UPF running on a combination of hardware switches and servers.

backend translator polls the corresponding hardware switch counters and relays the response.

CHAPTER

SIX

CHAPTER 6: MANAGED CLOUD SERVICE

This chapter describes how to assemble all the pieces described in the previous chapters to provide 5G connectivity as a managed cloud service. Such a service might be deployed in enterprises, for example, in support of collection of operational data, video, robots, IoT devices, and so on—a set of use cases sometimes referred to as Industry 4.0.

The first step is to implement all the components using cloud native building blocks. We start by introducing those building blocks in Section 6.1. The second step is to introduce yet another component—a *Cloud Management Platform*—that is responsible for operationalizing 5G-as-a-Service. The rest of the sections describe how to build such a management system using open source tools.

Before getting into the details, it is important to remember that mobile cellular service (both voice and broadband) has been offered as a Telco service for 40 years. Treating it as a managed cloud service is a significant departure from that history, most notably in how the connectivity it provides is operated and managed. As a consequence, the Cloud Management Platform described in this chapter is significantly different from the legacy OSS/BSS mechanisms that have traditionally been the centerpiece of the Telco management machinery. The terminology is also different, but that only matters if you are trying to map Telco terminology onto cloud terminology (which we are not). We take up the "terminology mapping problem" in a companion book, and here focus instead on a from-scratch cloud-based design.

Further Reading

L. Peterson, A. Bavier, S. Baker, Z. Williams, and B. Davie. Edge Cloud Operations: A Systems Approach. June 2022.

6.1 6.1 Building Blocks

The implementation strategy starts with commodity hardware and open source software. These building blocks will be familiar to anyone who has built a cloud native application, but they deserve to be explicitly named in a discussion of mobile cellular networks, which have historically been built using closed, proprietary hardware devices.

The hardware building blocks include bare-metal servers and switches, which might include ARM or x86 processor chips and Tomahawk or Tofino switching chips, respectively. A physical cloud cluster is then constructed with the hardware building blocks arranged as shown in Figure 6.1: one or more racks of servers connected by a leaf-spine switching fabric. We show the servers above the switching fabric to emphasize that software running on the servers controls the switches (as we will see in the next section).

The software building blocks start with the following open source components:

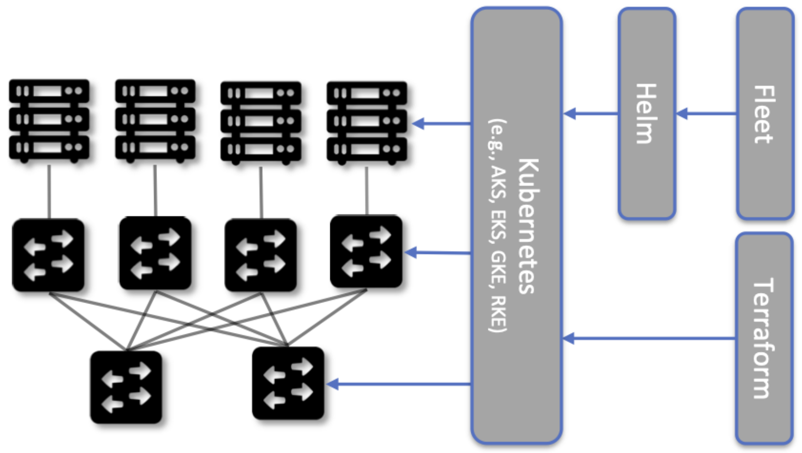

Figure 6.1.: Example building block components used to construct an edge cloud, including commodity servers and switches, interconnected by a leaf-spine switching fabric.

1. Docker containers package software functionality.
2. Kubernetes instantiates and interconnects a set of containers.
3. Helm specifies how collections of related containers are interconnected to build microservice-based applications.
4. Fleet specifies how a set of Kubernetes applications are to be deployed on the available infrastructure.
5. Terraform provisions a set of one or more Kubernetes clusters, configuring them to host microservice applications.

Docker is a container runtime that leverages OS isolation APIs to instantiate and run multiple containers, each of which is an instance defined by a Docker image. Docker images are most frequently built using a Dockerfile, which uses a layering approach that allows sharing and building customized images on top of base images. A final image for a particular task incorporates all dependencies required by the software that is to run in the container, resulting in a container image that is portable across servers, depending only on the kernel and Docker runtime. We also assume one or more image artifact repositories of Docker containers that we will want to deploy in our cloud, of which https://hub.docker.com/ is the best known example.

Further Reading

Docker Tutorial.

Kubernetes is a container orchestration system. It provides a programmatic interface for scaling container instances up and down, allocating server resources to them, setting up virtual networks to interconnect those instances, and opening service ports that external clients can use to access those instances. Behind the scenes, Kubernetes monitors the liveness of those containers, and automatically restarts any that have failed. In other words, if you instruct Kubernetes to spin up three instances of microservice X, Kubernetes will do its best to keep three instances of the container that implements X running at all times.

Further Reading

Kubernetes Tutorial.

Helm is a configuration manager that runs on top of Kubernetes. It issues calls against the Kubernetes API according to a developer-provided specification, known as a *Helm Chart*. It is now common practice for cloud applications built from a set of microservices to publish a Helm chart that defines how the application is to be deployed on a Kubernetes cluster. See https://artifacthub.io/ for a collection of publicly available Helm Charts.

Further Reading

Helm Tutorial.

Fleet, an application deployment manager, is responsible for installing a *Bundle* of Helm Charts on one or more target clusters. If we were trying to deploy a single Chart on just one Kubernetes cluster, then Helm would be sufficient. The value of Fleet is that it scales up that process, helping us manage the deployment of multiple charts across multiple clusters. Moreover, Fleet does this using an approach known as *Configuration-as-Code*, where the desired configuration is checked into a repo, just like any other software. Checking a new or updated Bundle into a repo triggers the deployment of the corresponding applications.

Further Reading

Fleet: GitOps at Scale.

Terraform is an infrastructure manager that, in our scenario, provisions one or more Kubernetes clusters, preparing them to host a collection of Helm-specified applications. It does this using an approach known as *Infrastructure-as-Code*, which documents exactly how the infrastructure is to be configured in a declarative format that can be (a) checked into a repo, (b) version-controlled, and (c) executed just like any piece of software. Terraform assumes an underlying provisioning API, with Microsoft's Azure Kubernetes Service (AKS), AWS's Amazon Elastic Kubernetes Service (EKS), Google's Google Kubernetes Engine (GKE) and Rancher's Rancher Kubernetes Engine (RKE) being widely available examples.

Further Reading

Terraform Tutorials.

The inter-related roles of Helm, Fleet, and Terraform can be confusing, in part because there is overlap in what each tries to do. One distinction is that Helm Charts are typically specified by *developers* as a way of specifying how an application is constructed from a set of microservices, whereas Fleet and Terraform give *operators* an opportunity to specify details of their particular deployment scenarios. A second distinction is that Helm and Fleet help manage the *applications running on* one or more Kubernetes clusters, whereas Terraform is used to set up and configure the *underlying Kubernetes clusters* in the first place. Again, there is overlap in the capabilities of these respective tools, but these two distinctions characterize how they are used in Aether. The more general takeaway is that cloud management has to accommodate both developers and operators, and to clearly delineate between applications and platforms.

6.2 6.2 Example Deployment

Using these building blocks, it is possible to construct a wide range of deployment scenarios for a managed 5G service. For illustrative purposes, we use a particular deployment based on the Aether edge cloud introduced in Chapter 2. Aether is an operational edge cloud that has been deployed to multiple sites, and most importantly for our purposes, includes an API that edge apps can use to customize 5G connectivity to better meet their objectives.

6.2.1 6.2.1 Edge Cloud

An Aether edge deployment, called ACE (Aether Connected Edge), is a Kubernetes-based cluster. It consists of one or more server racks interconnected by a leaf-spine switching fabric, with an SDN control plane (denoted SD-Fabric) managing the fabric. We briefly saw SD-Fabric in Chapter 5 as an implementation option for the Mobile Core's User Plane Function (UPF), but for an in-depth description of SD-Fabric, we refer you to a companion book.

Further Reading

L. Peterson, C. Cascone, B. O'Connor, T. Vachuska, and B. Davie. Software-Defined Networks: A Systems Approach. November 2021.

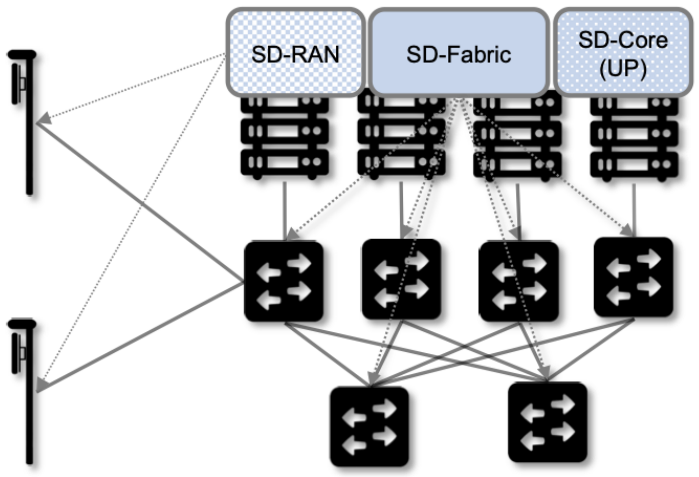

Figure 6.2.: Aether Connected Edge (ACE) = The cloud platform (Kubernetes and SD-Fabric) plus the 5G connectivity service (RAN and User Plane of Mobile Core). Dotted lines (e.g., between SD-RAN and the individual base stations, and between the Network OS and the individual switches) represent control relationships (e.g., SD-RAN controls the small cells and SD-Fabric controls the switches).

As shown in Figure 6.2, ACE hosts two additional microservice-based subsystems on top of this platform; they collectively implement *5G-as-a-Service*. The first subsystem, SD-RAN, is the SDN-based implementation of the Radio Access Network described in Chapter 4. It controls the small cell base stations deployed throughout the enterprise. The second subsystem, SD-Core, is an SDN-based implementation of the User Plane half of the Mobile Core described in Chapter 5. It is responsible for forwarding traffic between the RAN and the Internet. The SD-Core Control Plane (CP) runs off-site, and is not shown in Figure 6.2. Both subsystems (as well as the SD-Fabric), are deployed as a set of microservices, just as any other cloud native workload.

Once an edge cluster is running in this configuration, it is ready to host a collection of cloud-native edge applications (not shown in Figure 6.2). What's unique to our example configuration is its ability to connect such applications to mobile devices throughout the enterprise using the 5G Connectivity Service implemented by SD-RAN and SD-Core, without the resulting network traffic ever leaving the enterprise; a scenario known as *local breakout*. Moreover, this service is offered as a managed service, with enterprise system administrators able to use a programmatic API (and associated GUI portal) to control that service; that is, authorize devices, restrict access, set QoS profiles for different devices and applications, and so on.

6.2.2 6.2.2 Hybrid Cloud

While it is possible to instantiate a single ACE cluster in just one site, Aether is designed to support multiple edge deployments, all of which are managed from the central cloud. Such a hybrid cloud scenario is depicted in Figure 6.3, which shows two subsystems running in the central cloud: (1) one or more instantiations of the Mobile Core Control Plane (CP), and (2) the Aether Management Platform (AMP).

Each SD-Core CP controls one or more SD-Core UPFs. Exactly how CP instances (running centrally) are paired with UPF instances (running at the edges) is a runtime decision, and depends on the degree of isolation the enterprise sites require. AMP is Aether's realization of a Cloud Management Platform; it is responsible for managing all the centralized and edge subsystems (as introduced in the next section).

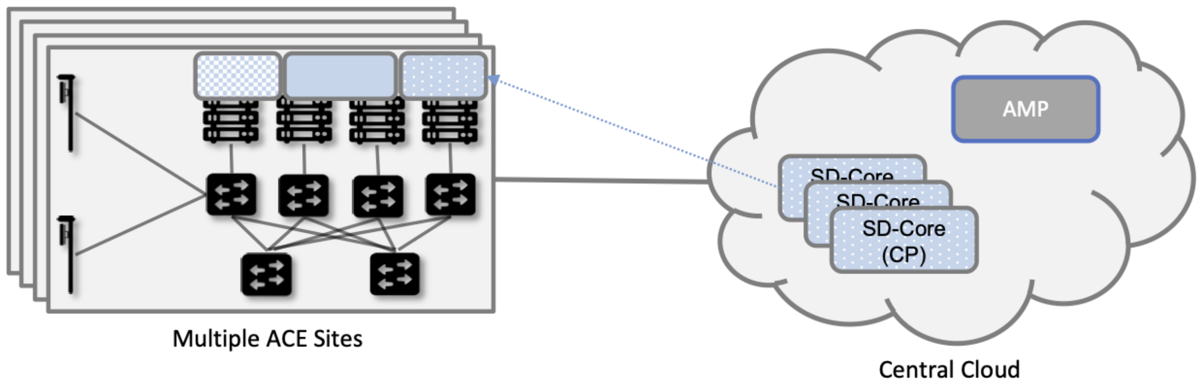

Figure 6.3.: Aether runs in a hybrid cloud configuration, with Control Plane of Mobile Core and the Aether Management Platform (AMP) running in the Central Cloud.

There is an important aspect of this hybrid cloud that is not obvious from Figure 6.3, which is that the "hybrid cloud" we keep referring to is best described as a set of Kubernetes clusters, rather than a set of physical clusters. This is because, while each ACE site usually corresponds to a physical cluster built out of bare-metal components, each of the SD-Core CP subsystems shown in Figure 6.3 is actually deployed in a logical Kubernetes cluster on a commodity cloud. The same is true for AMP. Aether's centralized components are able to run in Google Cloud Platform, Microsoft Azure, and Amazon's AWS. They also run as an emulated cluster implemented by a system like KIND (Kubernetes in Docker), making it possible for developers to run these components on their laptops.

Near-Edge vs Far-Edge

We use enterprises as the exemplar edge deployment in this book, without prescribing a role for traditional MNOs. When traditional MNOs are involved, it is not uncommon for them to make a distinction between the "near-edge" and the "far-edge", where the far-edge corresponds to the enterprise and the near-edge corresponds to their traditional aggregation points (or Central Offices), as described in

Section 1.2. In such a scenario, it is typically the case that the RU and DU are located at the far-edge (on-prem), while the CU—along with both the Control and User Planes of the Mobile Core—run in the near-edge. Such a configuration does not support local breakout, since all traffic must travel to the near-edge before being routed to the edge app (which might be running back in the enterprise).

In contrast, the deployment described in this Chapter has everything except the Mobile Core Control Plane (CP) running on-prem. Moreover, because there is no traditional MNO involved, there is no near-edge to speak of, with the Core CP instead running in a central cloud. For example, this section describes a deployment with SD-Core (CP) running in the Google Cloud. It is the case, however, that the SD-Core (CP) can optionally run on-prem if a fully local configuration is preferred. Where each component runs is a configuration option.

6.2.3 6.2.3 Stakeholders

With the understanding that our target environment is a collection of Kubernetes clusters—some running on bare-metal hardware at edge sites and some running in central datacenters—there is an orthogonal issue of how decision-making responsibility for those clusters is shared among multiple stakeholders. Identifying the relevant stakeholders is an important prerequisite for establishing a cloud service, and while the example we use may not be suitable for all situations, it does illustrate the design implications.

For Aether, we care about two primary stakeholders: (1) the *cloud operators* who manage the hybrid cloud as a whole, and (2) the *enterprise users* who decide on a per-site basis how to take advantage of the local cloud resources (e.g., what edge applications to run and how to slice connectivity resources among those apps). We sometimes call the latter "enterprise admins" to distinguish them from "end-users" who might want to manage their own personal devices.

Aether is multi-tenant in the sense that it authenticates and isolates these stakeholders, allowing each to access only those objects they are responsible for. This makes the approach agnostic as to whether all the edge sites belong to a single organization (with that organization also responsible for operating the cloud), or alternatively, there being a separate organization that offers a managed service to a set of distinct enterprises (each of which spans one or more sites).

There is a third stakeholder of note—third-party service providers—which points to the larger issue of how we deploy and manage the edge applications that take advantage of 5G-as-a-Service. The approach Aether adopts is to expect service providers to make their applications available either as source code (which works for open source or in-house apps), or as standard cloud native artifacts (e.g., Docker images and Helm charts). Either format can be fed into the Lifecycle Management pipeline described in Section 6.3.2. The alternative would be for edge service providers to share operational responsibility for the edge cloud with the cloud operator, which is possible if the infrastructure running at the edge is either multi-tenant or a multi-cloud.

6.2.4 6.2.4 Alternative Configurations

The deployment just described is Aether in its full glory. Simpler configurations are also possible, which makes sense in less demanding scenarios. Examples include:

- Small edge clusters can be built with only a single switch (or two switches for resiliency), with or without SDN-based control. In the limit, an Aether edge can run on a single server.
- It is possible to substitute legacy small cells for O-RAN compliant small cells and the SD-RAN solution that includes a near RT-RIC and associated xApps.

- It is possible co-locate both AMP and the SD-Core on the edge cluster, resulting in a complete Aether deployment that is self-contained in a single site.

These are all straightforward configuration options. A very different approach is to start with an edge cluster that is managed by one of the hyperscalers, rather than have Aether provision Kubernetes on bare-metal. Google's Anthos, Microsoft's Azure Arc, and Amazon's ECS-Anywhere are examples of such edge cloud products. In such a scenario, AMP still manages the SD-Core and SD-RAN applications running on top of Kubernetes, but not the underlying platform (which may or may not include an SDN-based switching fabric).

Another variable in how 5G can be deployed at the edge is related to who owns the underlying cloud infrastructure. Instead of a cloud provider, an enterprise, or a traditional MNO owning the hardware, there are situations where a third-party, often called a *neutral host*, owns and operates the hardware (along with the real estate it sits in), and then rents access to these resources to multiple 5G providers. Each mobile service provider is then a tenant of that shared infrastructure.

This kind of arrangement has existed for years, albeit with conventional RAN devices, but shifting to a cloud-based design makes it possible for neutral hosts to lease access to *virtualized* edge resources to their tenants. In principle, the only difference between this scenario and today's multi-tenant clouds is that such providers would offer edge resources—located in cell towers, apartment buildings, and dense urban centers—instead of datacenter resources. The business arrangements would also have to be different from Private 5G, but the technical design outlined in this book still applies.

6.3 6.3 Cloud Management Platform

Operationalizing the hardware and software components described in the previous two sections is the essence of what it means to offer 5G as a *managed service*. This responsibility falls to the Cloud Management Platform, which in Aether corresponds to the centralized AMP component shown in Figure 6.3. AMP manages both the distributed set of ACE clusters and one or more SD-Core CP clusters running in the central cloud.

The following uses AMP to illustrate how to deliver 5G-as-a-Service, but the approach generalizes because AMP is based on widely-used open source tools. For more details about all the subsystems involved in operationalizing an edge cloud, we refer you to the companion book mentioned in the introduction to this chapter.

At a high level, AMP is organized around the four subsystems shown in Figure 6.4:

- **Resource Provisioning** is responsible for initializing resources (e.g., servers, switches) that add, replace, or upgrade capacity. It configures and bootstraps both physical and virtual resources, bringing them up to a state so Lifecycle Management can take over and manage the software running on those resources.
- **Lifecycle Management** is responsible for continuous integration and deployment of the software components that collectively implement 5G-as-a-Service. It adopts the GitOps practice of *Configuration-as-Code*, using Helm Charts, Terraform Templates, and Fleet Bundles to specify how functionality is to be deployed and configured.
- **Service Orchestration** provides a means to manage services once they are operational. It defines an API that hides the implementation details of the underlying microservices, and is used to manage the provided 5G connectivity service.
- **Monitoring & Telemetry** is responsible for collecting, archiving, evaluating, and analyzing operational data generated by the underlying components. It makes it possible to diagnose and respond

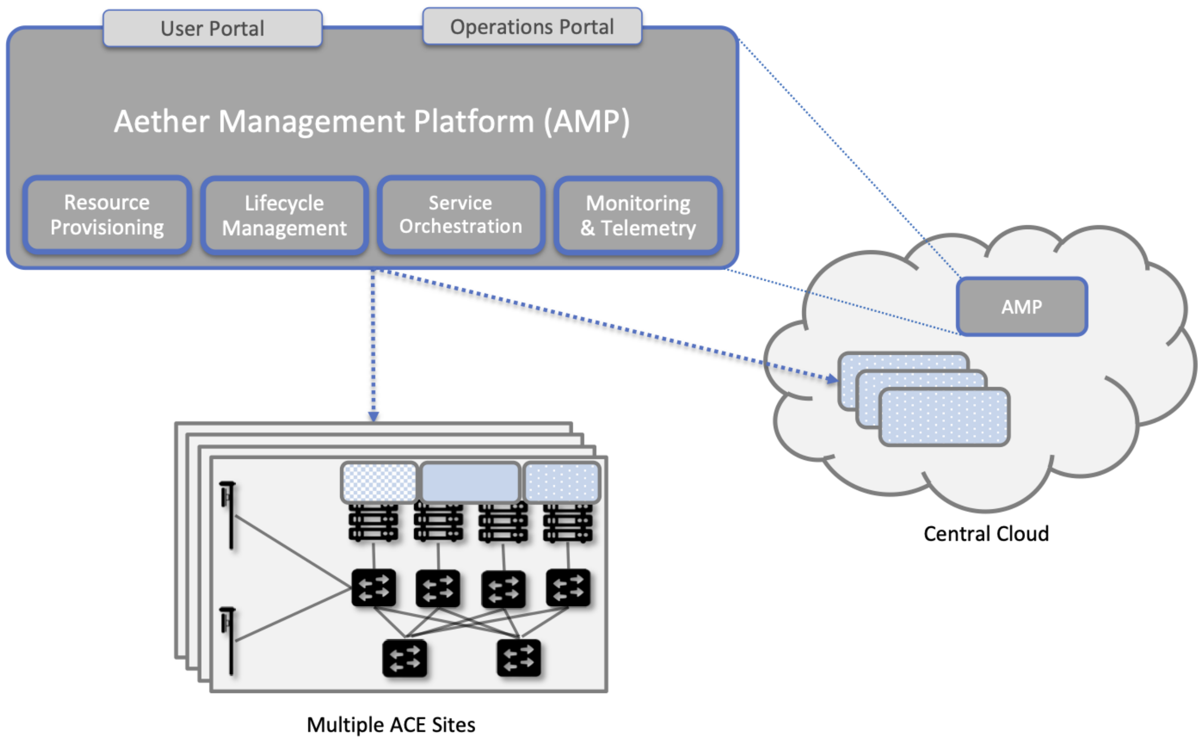

Figure 6.4.: The four subsystems that comprise AMP: Resource Provisioning, Lifecycle Management, Service Orchestrator, and Monitoring & Telemetry.

to failures, tune performance, do root cause analysis, perform security audits, and understand when it is necessary to provision additional capacity.

AMP implements all four subsystems, but an alternative perspective that characterizes the management platform as having *online* and *offline* components is also instructive. Such a two dimensional schematic is shown in Figure 6.5. Lifecycle Management (coupled with Resource Provisioning) runs offline, sitting adjacent to the hybrid cloud. Operators and Developers provision and change the system by checking code (including configuration specs) into a repo, which in turn triggers an upgrade of the running system. Service Orchestration (coupled with Monitoring and Telemetry) runs online, layered on top of the hybrid cloud being managed. It defines an API that can be used to read and write parameters of the running system, which serves as a foundation for building closed-loop control.

The offline and online aspects of cloud management are related in the sense that the offline component is also responsible for lifecycle-managing the online component. This is because the latter is deployed as a collection of Kubernetes applications, just like SD-Core and SD-RAN. Version management is a key aspect of this relationship since the runtime API to the 5G connectivity service has to stay in sync with the underlying implementation of the constituent subsystems. How Aether realizes version control is described in more detail in the companion Edge Cloud Operations book.

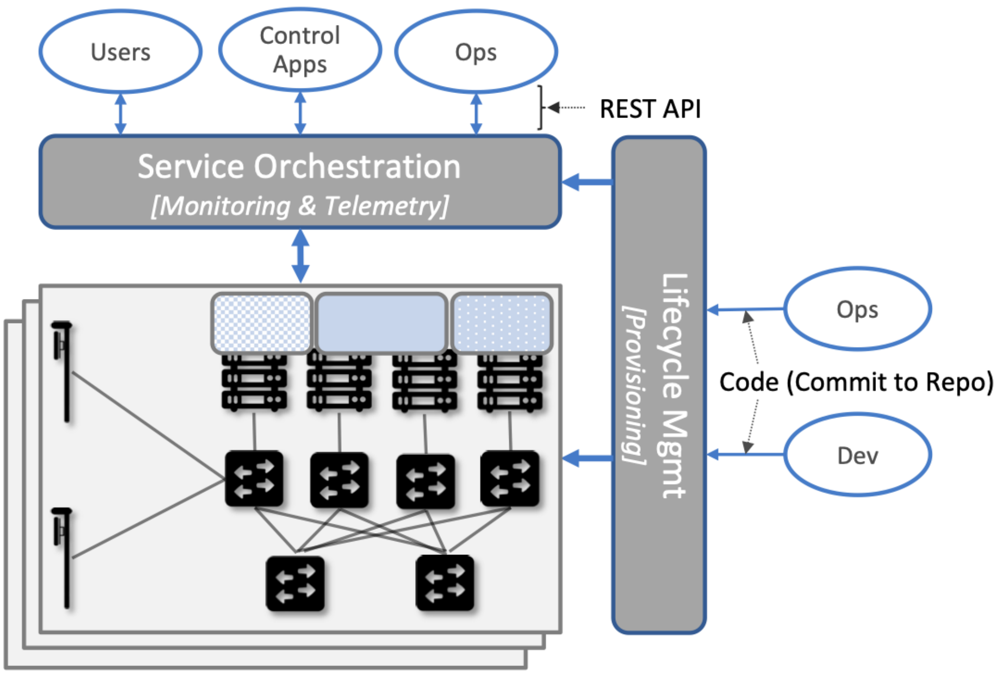

Figure 6.5.: Alternative representation of the management platform, highlighting the offline and online aspects of cloud management.

6.3.1 6.3.1 Resource Provisioning

Resource Provisioning is the process of bringing virtual and physical resources online. For physical resources, it has both a hands-on component (racking and connecting devices) and a bootstrap component (configuring how the resources boot into a “ready” state). When utilizing virtual resources (e.g., VMs instantiated on a commercial cloud) the “rack and connect” step is carried out by a sequence of API calls rather than a hands-on technician.

Because we want to automate the sequence of calls needed to activate virtual infrastructure, we adopt an approach known as *Infrastructure-as-Code*. This is where Terraform comes into play. The general idea is to document, in a declarative format that can be “executed”, exactly what our infrastructure is to look like. The code defines how the infrastructure is to be configured.

When a cloud is built from a combination of virtual and physical resources, as is the case for a hybrid cloud like Aether, we need a seamless way to accommodate both. To this end, our approach is to first overlay a *logical structure* on top of hardware resources, making them roughly equivalent to the virtual resources we get from a commercial cloud provider. This results in a hybrid scenario similar to the one shown in Figure 6.6. One way to think about this is that the task of booting hardware into the “ready” state involves installing and configuring several subsystems that collectively form the cloud platform. It is this platform that Terraform interacts with, indirectly, through a cloud provisioning API.

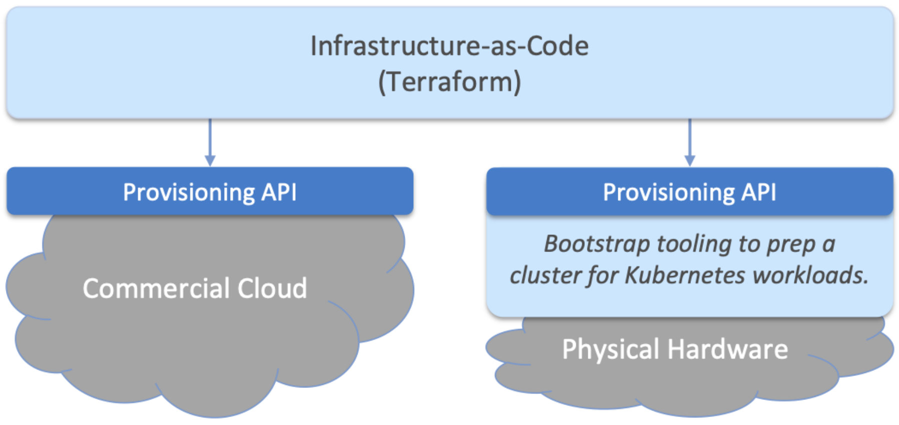

Figure 6.6.: Resource Provisioning in a hybrid cloud that includes both physical and virtual resources.

6.3.2 6.3.2 Lifecycle Management

Lifecycle Management is concerned with updating and evolving a running system over time. Figure 6.7 gives an overview of the pipeline/toolchain that make up the two halves of Lifecycle Management—Continuous Integration (CI) and Continuous Deployment (CD). The key thing to focus on is the Image and Config Repos in the middle. They represent the "interface" between the two halves: CI produces Docker Images and Helm Charts, storing them in the respective Repositories, while CD consumes Docker Images and Helm Charts, pulling them from the respective Repositories.

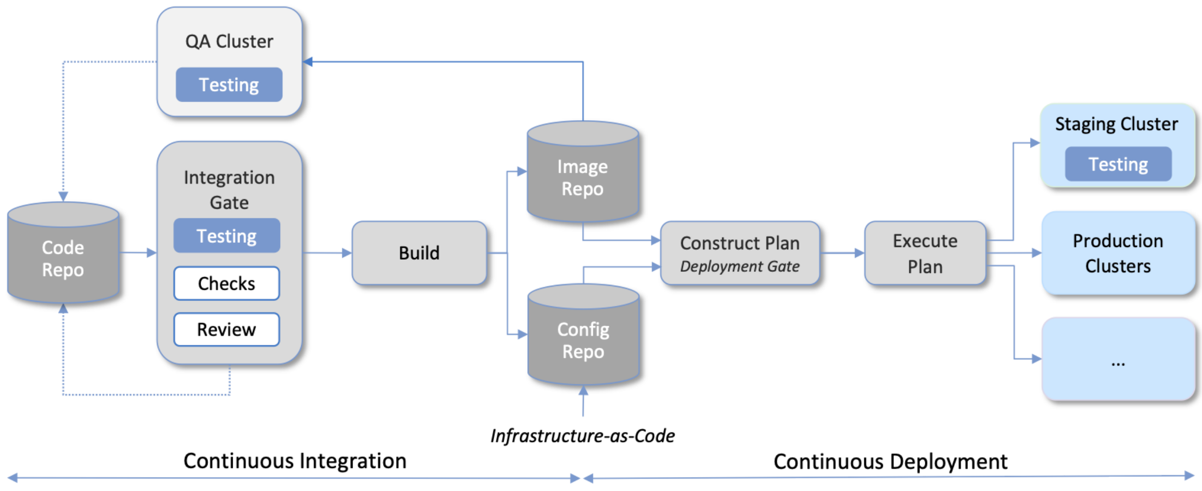

Figure 6.7.: Overview of the CI/CD pipeline.

The Config Repo also contains declarative specifications of the infrastructure artifacts (specifically, Terraform templates and Fleet Bundles). These files are input to Lifecycle Management, which implies that Terraform and Fleet gets invoked as part of CI/CD whenever these files change. In other words, CI/CD keeps both the software-related components in the underlying cloud platform and the microservice workloads that run on top of that platform up to date.

Continuous Delivery vs Deployment

You will also hear CD refer to "Continuous Delivery" instead of "Continuous Deployment", but we are interested in the complete end-to-end process, so CD will always imply the latter in this book. But *keep in mind that "continuous" does not necessarily mean "instantaneous"; there can be a variety of gating functions injected into the CI/CD pipeline to control when and how upgrades get rolled out. The important point is that all the stages in the pipeline are automated.*

So what exactly does "Continuous Delivery" mean? Arguably, it's redundant when coupled with "Continuous Integration" since the set of artifacts being produced by the CI half of the pipeline (e.g., Docker images) is precisely what's being delivered. There is no "next step" unless you also deploy those artifacts. It's hair-splitting, but some would argue CI is limited to testing new code and Continuous Delivery corresponds to the final "publish the artifact" step. For our purposes, we lump "publish the artifact" into the CI half of the pipeline.

There are three takeaways from this overview. The first is that by having well-defined artifacts passed between CI and CD (and between operators responsible for resource provisioning and CD), the subsystems are loosely coupled, and able to perform their respective tasks independently. The second is that all authoritative state needed to successfully build and deploy the system is contained within the pipeline, specifically, as declarative specifications in the Config Repo. This is the cornerstone of *Configuration-as-Code* (also known as *GitOps*), the cloud native approach to CI/CD. The third is that there is an opportunity for operators to apply discretion to the pipeline, as denoted by the *"Deployment Gate"* in the Figure, controlling what features get deployed when.

The third repository shown in Figure 6.7 is the Code Repo (on the far left). Developers continually check new features and bug fixes into this repo, which triggers the CI/CD pipeline. A set of tests and code reviews are run against these check-ins, with the output of those tests/reviews reported back to developers, who modify their patch sets accordingly. (These develop-and-test feedback loops are implied by the dotted lines in Figure 6.7.)

The far right of Figure 6.7 shows the set of deployment targets, with *Staging* and *Production* called out as two illustrative examples. The idea is that a new version of the software is deployed first to a set of Staging clusters, where it is subjected to realistic workloads for a period of time, and then rolled out to the Production clusters once the Staging deployments give us confidence that the upgrade is reliable.

Finally, two of the CI stages shown in Figure 6.7 identify a *Testing* component. One is a set of component-level tests that are run against each patch set checked into the Code Repo. These tests gate integration; fully merging a patch into the Code Repo requires first passing this preliminary round of tests. Once merged, the pipeline runs a build across all the components, and a second round of testing happens on a *Quality Assurance (QA)* cluster. Passing these tests gate deployment, but as just noted, testing also happens in the Staging clusters as part of the CD end of the pipeline.

6.3.3 6.3.3 Service Orchestration

Service Orchestration is responsible for managing the Kubernetes workloads once they are up and running, which in our case means providing a programmatic API that can be used by various stakeholders to manage the 5G connectivity service. As shown in Figure 6.8, the Service Orchestrator hides the implementation details of 5G connectivity, which spans four different components and multiple clouds. It does this by providing a coherent service interface for users, enabling them to authorize devices and set QoS parameters on an end-to-end basis.

In other words, the Service Orchestrator defines an abstraction layer on top of a collection of backend components, effectively turning them into an externally visible (and controllable) cloud service. In some situations a single backend component might implement the entirety of a service, but in the case of 5G, which is constructed from a collection of disaggregated components, Service Orchestration is where we define an API that logically integrates those components into a unified and coherent whole. It is

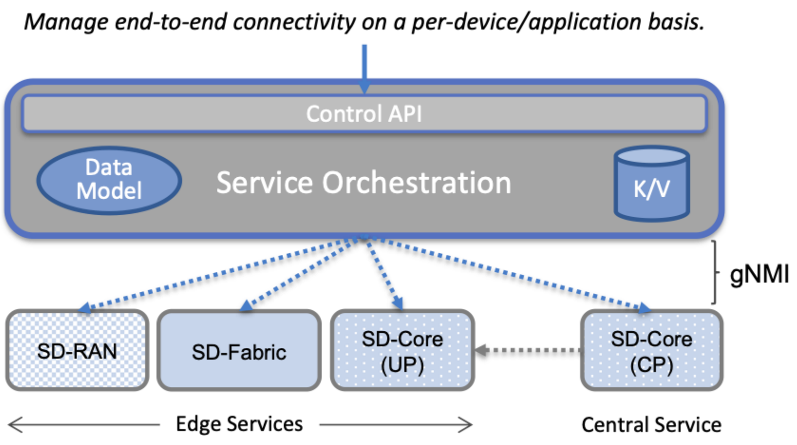

Figure 6.8.: Example use case that requires ongoing runtime control.

also an opportunity to "raise the level of abstraction" for the underlying subsystems, hiding unnecessary implementation details.

We describe this connectivity interface in Section 6.4. For now, our focus is on the main issues Service Orchestration must address in order to offer such an API. At a high level, it must:

1. Authenticate the principal wanting to perform the operation.
2. Determine if that principal has sufficient privilege to carry out the operation.
3. Push the new parameter setting(s) to one or more backend components.
4. Record the specified parameter setting(s), so the new value(s) persist.

Central to this role is the requirement that Service Orchestration be able to represent a set of abstract objects, which is to say, it implements a *data model*. The API is then generated from this data model, and persistent state associated with instances of the models is stored in a key-value store. Aether uses YANG to specify the models, in part because it is a rich language for data modeling, but also because there is a robust collection of YANG-based tools that we can build upon.

Further Reading

YANG - A Data Modeling Language for the Network Configuration Protocol. RFC 6020. October 2010.

Finally, changes to the model-defined parameters must be propagated to the backend components, and in practice there is no established API for doing this. Aether assumes gNMI as its southbound interface to communicate configuration changes to the software services, where an Adapter (not shown in the figure) has to be written for any services that do not support gNMI natively.

6.3.4 6.3.4 Monitoring and Telemetry

Collecting telemetry data for a running system is an essential function of the management platform. It enables operators to monitor system behavior, evaluate performance, make informed provisioning decisions, respond to failures, identify attacks, and diagnose problems. There are three types of telemetry data—*metrics*, *logs*, and *traces*—along with open source software stacks available to help collect, store, and act upon each of them.

Metrics are quantitative data about a system. These include common performance metrics such as link bandwidth, CPU utilization, and memory usage, but also binary results corresponding to “up” and “down”, as well as other state variables that can be encoded numerically. These values are produced and collected periodically (e.g., every few seconds), either by reading a counter, or by executing a runtime test that returns a value. These metrics can be associated with physical resources such as servers and switches, virtual resources such as VMs and containers, or high-level abstractions such as the *Connectivity Service* described in the next section. Given these many possible sources of data, the job of the metrics monitoring stack is to collect, archive, visualize, and optionally analyze this data. Prometheus is a popular open source tool for storing and querying metrics.

Further Reading

Prometheus.

Logs are the qualitative data that is generated whenever a noteworthy event occurs. This information can be used to identify problematic operating conditions (i.e., it may trigger an alert), but more commonly, it is used to troubleshoot problems after they have been detected. Various system components—all the way from the low-level OS kernel to high-level cloud services—write messages that adhere to a well-defined format to the log. These messages include a timestamp, which makes it possible for the logging stack to parse and correlate messages from different components. ElasticSearch is a widely-used tool for storing and analyzing log messages.

Further Reading

ElasticSearch.

Traces are a record of causal relationships (e.g., Service A calls Service B) resulting from user-initiated transactions or jobs. They are related to logs, but provide more specialized information about the context in which different events happen. Tracing is well understood in a single program, where an execution trace is commonly recorded as an in-memory call stack, but traces are inherently distributed across a graph of network-connected microservices in a cloud setting. This makes the problem challenging, but also critically important because it is often the case that the only way to understand time-dependent phenomena—such as why a particular resource is overloaded—is to understand how multiple independent workflows interact with each other. Jaeger is a popular open source tool used for tracing.

Further Reading

Jaeger: End-to-End Distributed Tracing.

Finally, note that our framing of monitoring and telemetry as part of the online aspect of management is somewhat simplistic. Certainly telemetry data is collected from online processes embedded in a running system, and such data can be coupled with online control operations to realize closed-loop control, but it is also the case that some telemetry data is evaluated offline. This is true for logs and traces used to diagnose problems, and for performance data used to make provisioning decisions, both of which can lead to code changes that feed back into the next iteration of lifecycle management.

6.4 6.4 Connectivity API

The visible aspect of a 5G service is the programmatic interface it provides to users, giving them the ability to control and customize the underlying connectivity service. This API is implemented by the Service Orchestrator outlined in the previous section, but what we really care about is the interface itself. Using Aether as a concrete example, this section describes such an API.

Like many cloud services, the API for 5G-as-a-Service is RESTful. This means it supports REST's GET, POST, PATCH, and DELETE operations on a set of resources (objects):

- GET: Retrieve an object.
- POST: Create an object.
- PUT, PATCH: Modify an existing object.
- DELETE: Delete an object.

Each object, in turn, is typically defined by a data model. In Aether this model is specified in YANG, but rather than dive into the particulars of YANG, this section describes the models informally by describing the relevant fields.

Every object contains an *id* field that is used to uniquely identify the object. Some objects contain references to other objects. For example, many objects contain references to the *Enterprise* object, which allows them to be associated with a particular enterprise. That is, references are constructed using the *id* field of the referenced object.

In addition to the *id* field, several other fields are also common to all models. These include:

- *description*: A human-readable description, used to store additional context about the object.
- *display-name*: A human-readable name that is shown in the GUI.

As these fields are common to all models, we omit them from the per-model descriptions that follow. Note that we use upper case to denote a model (e.g., *Enterprise*) and lower case to denote a field within a model (e.g., *enterprise*).

6.4.1 6.4.1 Enterprises

Aether is deployed in enterprises, and so needs to define a representative set of organizational abstractions. These include *Enterprise*, which forms the root of a customer-specific hierarchy. The *Enterprise* model is referenced by many other objects, and allows those objects to be scoped to a particular Enterprise for ownership and role-based access control purposes. *Enterprise* contains the following fields:

- *connectivity-service*: A list of backend subsystems that implement connectivity for this enterprise. This list corresponds to the API endpoint for the SD-Core, SD-Fabric, and SD-RAN components.

Enterprises are further divided into *Sites*. A site is a point-of-presence for an *Enterprise* and may be either physical or logical (i.e., a single geographic location could contain several logical sites). The *Site* model, in turn, contains the following fields:

- *enterprise*: A link to the *Enterprise* that owns this site.

- *imsi-definition*: A description of how IMSIs are constructed for this site. It consists of the following sub-fields:
  - *mcc*: Mobile country code.
  - *mnc*: Mobile network code.
  - *enterprise*: A numeric enterprise id.
  - *format*: A mask that defines how the above three fields are encoded in an IMSI. For example *CCCNNNEEESSSSSS* specifies an IMSI with a 3-digit MCC, a 3-digit MNC, a 3-digit ENT, and a 6-digit subscriber.

As a reminder, an IMSI is burned into every SIM card, and is used to identify and locate UEs throughout the global cellular network.

6.4.2 6.4.2 Slices

Aether models 5G connectivity as a *Slice*, which represents an isolated communication channel (and associated QoS parameters) that connects a set of devices (modeled as a *Device-Group*) to a set of applications (each of which is modeled as an *Application*). For example, an enterprise might configure one slice to carry IoT traffic and another slice to carry video traffic. The *Slice* model has the following fields:

- *device-group*: A list of *Device-Group* objects that can participate in this *Slice*. Each entry in the list contains both the reference to the *Device-Group* as well as an *enable* field which may be used to temporarily remove access to the group.
- *application*: A list of *Application* objects that are either allowed or denied for this *Slice*. Each entry in the list contains both a reference to the *Application* as well as an *allow* field which can be set to *true* to allow the application or *false* to deny it.
- *template*: Reference to the *Template* that was used to initialize this *Slice*.
- *upf*: Reference to the User Plane Function (*UPF*) that should be used to process packets for this *Slice*. Multiple *Slices* may share a single *UPF*.
- *enterprise*: Reference to the *Enterprise* that owns this *Slice*.
- *site*: Reference to the *Site* where this *Slice* is deployed.
- *sst*, *sd*: 3GPP-defined slice identifiers assigned by the operations team.
- *mbr.uplink*, *mbr.downlink*, *mbr.uplink-burst-size*, *mbr.downlink-burst-size*: Maximum bit-rate and burst sizes for this slice.

The rate-related parameters are initialized using a selected *template*, as described below, but these values may be changed at runtime. Also note that this example illustrates how modeling can be used to enforce invariants, in this case, that the *Site* of the *UPF* and *Device-Group* must match the *Site* of the *Slice*. That is, the physical devices that connect to a slice and the UPF that implements the core segment of the slice must be constrained to a single physical location.

At one end of a Slice is a *Device-Group*, which identifies a set of devices that are allowed to use the Slice to connect to various applications. The *Device-Group* model contains the following fields:

- *imsis*: A list of IMSI ranges. Each range has the following fields:
  - *name*: Name of the range. Used as a key.
  - *imsi-range-from*: First IMSI in the range.

   - *imsi-range-to*: Last IMSI in the range. Can be omitted if the range only contains one IMSI.

- *ip-domain*: Reference to an *IP-Domain* object that describes the IP and DNS settings for UEs within this group.
- *site*: Reference to the site where this *Device-Group* may be used. (This field indirectly identifies the *Enterprise* since a *Site* contains a reference to *Enterprise*.)
- *mbr.uplink*, *mbr.downlink*: Maximum bit-rate for the device group.
- *traffic-class*: The traffic class to be used for devices in this group.

At the other end of a Slice is a list of *Application* objects, which specifies the endpoints for the program devices talk to. The *Application* model contains the following fields:

- *address*: The DNS name or IP address of the endpoint.
- *endpoint*: A list of endpoints. Each has the following fields:
   - *name*: Name of the endpoint. Used as a key.
   - *port-start*: Starting port number.
   - *port-end*: Ending port number.
   - *protocol*: Protocol (*TCP|UDP*) for the endpoint.
   - *mbr.uplink*, *mbr.downlink*: Maximum bitrate for devices communicating with this application.
   - *traffic-class*: Traffic class for devices communicating with this application.
- *enterprise*: Link to an *Enterprise* object that owns this application. May be left empty to indicate a global application that may be used by multiple enterprises.

Note that Aether's *Slice* abstraction is similar to 3GPP's specification of a "slice", but the *Slice* model includes a combination of 3GPP-specified identifiers (e.g., *sst* and *sd*), and details about the underlying implementation (e.g., *upf* denotes the UPF implementation for the Core's user plane). The *Slice* model also includes fields related to RAN slicing, with the Service Orchestrator responsible for stitching together end-to-end connectivity across the RAN, Core, and Fabric.

6.4.3 6.4.3 QoS Profiles

Associated with each Slice is a QoS-related profile that governs how traffic carried by that slice is to be treated. This starts with a *Template* model, which defines the valid (accepted) connectivity settings. The Aether Operations team is responsible for defining these (the features they offer must be supported by the backend subsystems), with enterprises selecting the template they want applied to any instances of the connectivity service they create (e.g., via a drop-down menu). That is, templates are used to initialize *Slice* objects. The *Template* model has the following fields:

- *sst*, *sd*: Slice identifiers, as specified by 3GPP.
- *mbr.uplink*, *mbr.downlink*: Maximum uplink and downlink bandwidth.
- *mbr.uplink-burst-size*, *mbr.downlink-burst-size*: Maximum burst size.
- *traffic-class*: Link to a *Traffic-Class* object that describes the type of traffic.

You will see that the *Device-Group* and *Application* models include similar fields. The idea is that QoS parameters are established for the slice as a whole (based on the selected *Template*) and then individual devices and applications connected to that slice can define their own, more-restrictive QoS parameters on an instance-by-instance basis.

Finally, the *Traffic-Class* model specifies the classes of traffic, and includes the following fields:

- *arp*: Allocation and retention priority.
- *qci*: QoS class identifier.
- *pelr*: Packet error loss rate.
- *pdb*: Packet delay budget.

6.4.4 6.4.4 Other Models

The above description references other models, which we do not fully describe here. They include *AP-List*, which specifies a list of access points (radios); *IP-Domain*, which specifies IP and DNS settings; and *UPF*, which specifies the User Plane Function (the data plane element of the SD-Core) that is to forward packets on behalf of this particular instance of the connectivity service. The *UPF* model is necessary because Aether supports two different implementations: one runs as a microservice on a server and the other runs as a P4 program loaded into the switching fabric. Both implementations are described in Chapter 5.

CHAPTER SEVEN

ABOUT THE BOOK

This book incorporates introductory and background content from:

Title: 5G Mobile Networks: A Systems Approach
Authors: Larry Peterson and Oguz Sunay
Source: https://github.com/SystemsApproach/5g
License: CC BY-NC-ND 4.0

a version of which was also published by Morgan & Claypool in 2020 as part of their Synthesis Lectures on Network Systems.

7.1 Read the Book

This book is part of the Systems Approach Series, with an online version published at https://5G.systemsapproach.org.

For those users looking for our earlier 5G book, *5G Mobile Networks: A Systems Approach*, published as this URL, you can still find source for it archived on GitHub. This new book incorporates the background material covered in the old one, plus goes into significant detail about how Private 5G is implemented and deployed as a managed cloud service.

To track progress and receive notices about new versions, you can follow the project on Facebook and Mastodon. To read a running commentary on how the Internet is evolving, follow the Systems Approach on Substack.

7.2 Build the Book

To build a web-viewable version, you first need to download the source:

```
$ mkdir ~/systemsapproach
$ cd ~/systemsapproach
$ git clone https://github.com/SystemsApproach/private5g.git
$ cd private5g
```

The build process is stored in the Makefile and requires Python be installed. The Makefile will create a virtualenv (`venv-docs`) which installs the documentation generation toolset. You may also need to install the `enchant` C library using your system's package manager for the spelling checker to function properly.

To generate HTML in `_build/html`, run `make html`.

To check the formatting of the book, run `make lint`.

To check spelling, run `make spelling`. If there are additional words, names, or acronyms that are correctly spelled but not in the dictionary, please add them to the `dict.txt` file.

To see the other available output formats, run `make`.

7.3 Contribute to the Book

We hope that if you use this material, you are also willing to contribute back to it. If you are new to open source, you might check out this How to Contribute to Open Source guide. Among other things, you'll learn about posting *Issues* that you'd like to see addressed, and issuing *Pull Requests* to merge your improvements back into GitHub.

If you'd like to contribute and are looking for something that needs attention, see the wiki for the current TODO list.

CHAPTER

EIGHT

ABOUT THE AUTHORS

Larry Peterson is the Robert E. Kahn Professor of Computer Science, Emeritus at Princeton University, where he served as Chair from 2003-2009. His research focuses on the design, implementation, and operation of Internet-scale distributed systems, including the widely used PlanetLab and MeasurementLab platforms. He is currently contributing to the Aether access-edge cloud project at the Open Networking Foundation (ONF), where he serves as Chief Scientist. Peterson is a member of the National Academy of Engineering, a Fellow of the ACM and the IEEE, the 2010 recipient of the IEEE Kobayashi Computer and Communication Award, and the 2013 recipient of the ACM SIGCOMM Award. He received his Ph.D. degree from Purdue University.

Oguz Sunay is Chief Architect for the Network and Edge Group (NEX) at Intel, which he joined as part of Intel's acquisition of the Open Networking Foundation (ONF) engineering team. While at ONF, he was Vice President for Research & Development, where led all mobile-related projects. Before joining ONF, Sunay was the CTO at Argela-USA, where he was the innovator of a Programmable Radio Access Network Architecture (ProgRAN) for 5G that enabled the world's first dynamically programmable RAN slicing solution. He has also held prior industry positions at Nokia Research Center and Bell Laboratories, where he focused on 3G and 4G end-to-end systems architectures and participated and chaired various standardization activities. Sunay has also spent over 10 years in academia, as a Professor of Electrical and Computer Engineering. He holds many U.S. and European patents on various aspects of 3G, 4G, and 5G, and has authored numerous journal and conference publications. He received his Ph.D. and M.Sc. from Queen's University, Canada, and his B.Sc.Hon. from METU, Turkey.

Bruce Davie is a computer scientist noted for his contributions to the field of networking. He is a former VP and CTO for the Asia Pacific region at VMware. He joined VMware during the acquisition of Software Defined Networking (SDN) startup Nicira. Prior to that, he was a Fellow at Cisco Systems, leading a team of architects responsible for Multiprotocol Label Switching (MPLS). Davie has over 30 years of networking industry experience and has co-authored 17 RFCs. He was recognized as an ACM Fellow in 2009 and chaired ACM SIGCOMM from 2009 to 2013. He was also a visiting lecturer at the Massachusetts Institute of Technology for five years. Davie is the author of multiple books and the holder of more than 40 U.S. Patents.