1ST EDITION

Modern Network Observability

A hands-on approach using open source tools such as Telegraf, Prometheus, and Grafana

DAVID FLORES | CHRISTIAN ADELL
JOSH VANDERAA

Forewords by Eric Chou, Host, Network Automation Nerds Podcast, and Damien Garros, Infrastructure automation and observability architect, Co-Founder and CEO, OpsMill

Modern Network Observability

A hands-on approach using open source tools such as Telegraf, Prometheus, and Grafana

David Flores

Christian Adell

Josh VanDeraa

‹packt›

Modern Network Observability

Group Product Manager: Pavan Ramchandani
Publishing Product Manager: Prachi Sawant
Book Project Manager: Ashwin Kharwa
Senior Editor: Runcil Rebello
Technical Editor: Yash Bhanushali
Copy Editor: Safis Editing
Proofreader: Runcil Rebello
Indexer: Rekha Nair
Production Designer: Gokul Raj S.T
DevRel Marketing Coordinator: Marylou De Mello

First published: October 2024

Production reference:1120924

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK

ISBN 978-1-83508-106-8

www.packtpub.com

To my wife, Yaquelin, for your unwavering support and patience, especially during the writing of this book. To my family, for your endless love and encouragement. And to my friends and colleagues, for inspiring me and pushing me to finally discuss networks and observability—thank you!

- David Flores

To everyone who I have worked or collaborated with during my career. You have been instrumental in allowing me to satisfy my neverending curiosity.

- Christian Adell

To my terrific family, thank you for your support, love, and connection. This book shows my kids that you can do anything that you set your mind to. To those who have been part of my growth from youth through various stages of my life and career, I know I could not be here and do things like this without the path I have taken with you.

- Josh VanDeraa

Foreword 1

Imagine cruising down a winding road at midnight without turning on your headlights. Sure, you might enjoy the excitement and the adrenaline rush, but it is only a matter of time before you end up in an ambulance or worse. In the world of networking, operating a network without monitoring and observability is the equivalent of driving at night without headlights. We all understand the importance of visibility and insights into our networks, but for many years, we have relied on rudimentary tools such as ping and traceroute. Those tools were invented over 30 years ago when the network was much, much simpler. We desperately need to gain visibility into our networks, but we face the challenges of data collection, data processing, storage, scalability, vendor incompatibilities, visualization, and the list goes on. Many advancements have been made in recent years to address those challenges, such as streaming telemetry, accurate sampling of flow data, collecting and storing data at scale, and accurate visualization. However, we lack a comprehensive, non-biased form to illustrate how the individual pieces fit into the architecture and how they can be applied in a modern network.

I first met David and Christian while working at Network to Code and have known Josh for years before then. They were all brilliant engineers, but what impressed me the most beyond their technical abilities was their passion for helping others and empowering the community to succeed. In this book, they combine their deep knowledge of the world of modern network observability with years of practical, battle-tested experience. When I say operational experience, I mean the type that is wake-up-at-2 AM mission critical, leaving the scars that are beer-discussion and conference-talk worthy that normally require admission fees to hear about.

The book addresses the complex world of network observability challenges from architectural concepts to practical configuration examples with a focus on vendor-neutral, open source tools. You will gain the necessary knowledge, from collecting data in various forms, normalizing and enriching it, to visualizing and gaining the necessary insights to help you do your job, all with open source tools. If that sounds too good to be true, it is not. You can even take the architecture and tools introduced in the book and swap out the elements with other tools. That is the beauty of the vendor-neutral, open source tools introduced in this book. I believe the benefits you will gain from the first few chapters will outweigh the investment you've made. Better yet, the knowledge will pay dividends for years to come. I have been waiting for a book of this kind for years, and David, Christian, and Josh have done a masterful job in the delivery.

What are you waiting for? Let's dive into the modern world of network observability!

Eric Chou

Network Automation Advocate, Network to Code

Host, Network Automation Nerds Podcast

Founder, Network Automation Nerds, LLC.

Foreword 2

Observability has revolutionized how we manage infrastructure in the DevOps and cloud era, but its application to networking is still catching up. While network observability shares many similarities with broader infrastructure observability, there are specific challenges that have slowed its adoption in the networking field.

In this book, David, Christian, and Josh have done an outstanding job of covering everything network engineers and network reliability engineers need to know about network observability. They guide you from basic concepts to advanced examples, providing a comprehensive framework that has been successfully used in various production environments.

What makes this book special, in my view, is how network observability is presented not as an isolated component but as part of a larger automation stack. In observability, the quality of the data is everything. One of the benefits of modern observability is the ability to capture additional context, allowing us to slice and dice the data until we extract the most valuable information.

One of the key differences between monitoring and observability is the shift from a vertically integrated stack with limited flexibility to an extensible stack where users are encouraged to explore data and extract valuable insights to improve infrastructure reliability. This shift is important because it requires network engineers to develop new skills in data analysis.

At first, the amount of information may seem overwhelming, but it's important to remember that in a production environment, multiple roles contribute to defining and implementing observability solutions. Network reliability engineers are responsible for building the platform, while network engineers are the main users. However, network engineers should also take the time to understand how the platform works, as the quality of the data collected is crucial to the effectiveness of the observability stack.

A network engineer with a deep understanding of the types of data that can be collected from the infrastructure, and the context in which it is collected, will be an invaluable partner to the network reliability team in building the best possible platform. Whether you are a network engineer or a network reliability engineer, this book covers all the skills and technologies needed for both roles.

As you dive into this book and begin your journey, remember that the tools you use are just one part of the equation. It's more important to focus on understanding the core concepts and functional building blocks that form the foundation of effective observability practices. Tools may change and evolve, but the principles you'll learn here will remain relevant.

Don't be intimidated by the sheer volume of new information. It's natural to feel overwhelmed at first, but with time and practice, these concepts will become second nature. Approach this learning process with curiosity and an open mind, and you'll soon find yourself confidently navigating the exciting world of network observability.

For those new to this field, I recommend starting with a high-level understanding, which will help you identify which topics require deeper exploration and which ones are less critical for your role.

Over the years, I've had the opportunity to help many people discover and learn about network observability, and I've yet to see anyone who has invested time into it who wasn't genuinely excited by the new capabilities that come with these platforms.

Welcome to the exciting field of network observability.

Damien Garros

Infrastructure automation and observability architect

Co-Founder and CEO, OpsMill

Contributors

About the authors

David Flores is passionate about solving complex problems in network infrastructure, software architectures, automation, and observability. With experience with service providers, cloud providers, and system integrators, David has gained expertise in managing, automating, and building observability stacks for network infrastructure. Currently at CoreWeave, he focuses on enhancing automation and observability. David has also contributed to open source projects such as gns3fy, and actively shares his knowledge through blogs, workshops, and technical events. David is always curious and eager to keep himself updated and open to new ideas in the field.

Christian Adell is a principal architect at Network to Code He is focused on building network automation solutions for diverse use cases, with great emphasis on open source software. He is passionate about learning and helping others to grow, but also has more hobbies than hours in the day, so working remotely from Barcelona gives him the time and the space to achieve his dreams. Christian is a co-author of O'Reilly's *Network Programmability and Automation* book and a co-author of *Network Automation with Nautobot* by Packt. Also in relation to sharing knowledge, he is the organizer of the NetBCN community in Barcelona and has been collaborating with several universities for almost 20 years.

Josh VanDeraa is a network engineer and automation leader. Currently, he is a services director at Network to Code, driving value from network automation solutions. Josh has experience in automation and networking across retail, transportation, and managed services. In his free time, he enjoys being with his family or the Minnesota seasons. Josh co-authored *Network Automation with Nautobot* and self-published *Open Source Network Management*.

About the reviewers

Brad Haas is a seasoned professional in network automation, serving as vice president of professional services at Network to Code. With over two decades of experience in the field, Brad leads high-performing teams in complex, customer-focused projects, emphasizing a blend of technical skill and strategic insight. He advocates a data-informed approach to ensure technology aligns with business goals. His career features numerous technical certifications, including multiple CCIEs and cloud credentials. Previously, Brad contributed significantly to network operations and the adaptation of cloud-native applications and microservices, using technology to drive organizational transformation.

Thank you to my wife and children for your patience and love as I ventured into this technical review adventure. While it was fun for me reviewing chapters and testing labs, I know it meant more solo puzzles and one less player for game night at home. Thanks for not hiding my laptop, and for all the laughter and support that kept me going! Love you guys - Always.

Suhaib Saeed is a cloud network engineer with many years of experience in network design, automation, and observability and is currently focused on all things AWS. He has spent most of his career at various ISPs such as BT, working on large-scale projects for FTSE 100 clients. His current role is at Samsara, an IoT company on a mission to improve the safety, efficiency, and sustainability of the operations that power the global economy. Suhaib holds a BSc in Computer Networks and has authored multiple blogs on network automation.

Table of Contents

3

Data's Role in Network Observability 29

Part 2: Building an Effective Observability Stack

4

Observability Stack Architecture 73

5

Data Collectors 91

9

Alerting – Network Monitoring and Incident Management 285

10

Real-World Observability Architectures 317

Part 3: Using Your Network Observability Data

11

Applications of Your Observability Data – Driving Business Success 333

Preface

Modern Network Observability explores the current technology landscape for understanding the network operational state. Network monitoring tools have been around for many years, providing insights into how the network is performing. But, in today's world, the need for combining information (such as model-driven telemetry and with traditional SNMP monitoring) has brought about new tooling for observing networks that have not been seen before.

In the book, we will take you through some of the reasons for using the tooling, how to implement the tooling, and what you can do to integrate with other systems, such as modern AI and ML techniques.

Who this book is for

This book is for network analysts, network administrators, network architects, support professionals, engineers, and other IT professionals focused on networking. Leaders of networking organizations will find value in understanding capabilities and how these new capabilities may help to observe their networks.

What this book covers

Chapter 1, *Introduction to Monitoring and Observability*, introduces what monitoring and observability are for networking nowadays.

Chapter 2, *Role of Monitoring and Observability in Network Infrastructure*, introduces what the role of the data is and covers how data can provide service levels.

Chapter 3, *Data's Role in Network Observability*, covers the various types of data within the observability stack and how the data is gathered.

Chapter 4, *Observability Stack Architecture*, introduces a modern observability stack for network observability and how to work with data for a multi-vendor solution in a flexible way.

Chapter 5, *Data Collectors*, covers how to collect data for metrics and logs.

Chapter 6, *Data Distribution and Processing*, covers how to enrich data collection with metadata in the metrics pipeline and a method of scaling out data collection.

Chapter 7, *Data Storage Solutions for Network Observability*, covers the database and storage mechanisms for persisting metrics and logs.

Chapter 8, *Visualization – Bringing Network Observability to Life*, walks us through using visualization tools to bring dashboards and data out of the database and to your eyes.

Chapter 9, *Alerting – Network Monitoring and Incident Management*, covers the basics of alerting: getting actionable alerts based on the telemetry data that is collected.

Chapter 10, *Real-World Observability Architectures*, explains how to use the new information to set you up for success with the data.

Chapter 11, *Applications of Your Observability Data – Driving Business Success*, introduces how to start to use the data and how to tailor the solutions that you would build to those who need to see the data.

Chapter 12, *Automation Powered by Observability Data – Streamlining Network Operations*, brings to light the capabilities of integrating the tooling with other systems, with explanations of accessing data programmatically and how to build closed-loop and event-driven automation.

Chapter 13, *Leveraging Artificial Intelligence for Enhanced Network Observability*, completes the story by connecting the data within the tooling introduced to machine learning and artificial intelligence to supercharge your automation.

Appendix A sets up the lab environment and contains elaboration on how to use its components.

To get the most out of this book

This book assumes that you have a basic understanding of networking and of Linux shell fundamentals. There will be installation of applications on a Linux command line. Most of the command-line examples provided in the book are executed within a containerized environment, or native to most Linux shells. Some basic Python understanding is recommended as well, but not required.

Software/hardware covered in the book	**Operating system requirements**
Python	Windows Subsystem for Linux (WSL2), macOS, or Linux
Docker	WSL2, macOS, or Linux
Telegraf	WSL2, macOS, or Linux
Kafka	WSL2, macOS, or Linux
Prometheus	WSL2, macOS, or Linux
Grafana	WSL2, macOS, or Linux
ContainerLab	WSL2, macOS, or Linux

There may be some tools that are referenced using a cloud service for ease of gaining experience with the tool. There are no financial requirement to complete the examples provided throughout the book.

Another option is to execute the environments free within the open source community installations. These will be covered in more depth within the relevant chapters.

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Modern-Network-Observability`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The `anomaly` column gets the value `1` if it's outside of the `yhat` boundary."

A block of code is set as follows:

```
combined_results = pd.merge(
   current_metric,
   forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']],
   on='ds'
)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
    # Convert the `seconds` representation into datetime object
    data_dict['ds'] = pd.to_datetime(data[0], unit='s')
    # Save the interface counters value retrieved in a `float`
    format
    data_dict['y'] = float(data[1])
```

```
        metric_list.append(data_dict)
    df_metric = pd.DataFrame(metric_list)
```

Any command-line input or output is written as follows:

```
>>> pkts = sniff(filter="icmp and host 1.1.1.1", count=2)
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Click the **Settings** menu and then **Security**."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *Modern Network Observability*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/978-1-83508-106-8`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1: Understanding Monitoring and Observability

The first part of the book introduces you to monitoring and observability, taking us from where network observability began to where it is now. After reviewing where we have been and where monitoring and observability are going, we dive into the role of monitoring in organizations. The first part of the book wraps up with the different data types, such as logs, metrics, and traces, and how those types fit into the modern observability stack.

This part contains the following chapters:

- *Chapter 1*, *Introduction to Monitoring and Observability*
- *Chapter 2*, *Role of Monitoring and Observability in Network Infrastructure*
- *Chapter 3*, *Data's Role in Network Observability*

1

Introduction to Monitoring and Observability

Since the early days of computer networks, we have needed to detect failures on the different network components (e.g., hardware interface issues, cable cuts, or web service down) to determine outages that require corrective actions. This field has been known as **network monitoring**.

Interestingly, the last decade has witnessed numerous innovations in the field, especially related to new tools and practices around the DevOps culture. This culture emphasizes merging development and operations responsibilities requiring a better understanding of the operational state. Moreover, there has been a significant adoption of network automation. This advancement drives network operations, transforming monitoring from a passive component to an enabler of closed-loop processes. These changes have been the main drivers behind the evolution from network monitoring to **network observability**, and this book wants to help you understand and apply it to improve your network operations.

> **Note**
>
> Network observability is a broader topic, especially since the rise of running network applications directly in the host with technologies such as **extended Berkeley Packet Filter** (**eBPF**) and **Data Plane Development Kit** (**DPDK**). This kind of observability is not covered in detail in the book, even though most of the concepts are applicable too.

In this book, you will begin understanding the basics concepts related to network observability, and then, for the majority of it, we will explain how to build a modern network observability stack, with a practical, but not limited, emphasis on the **Telegraf** (`https://github.com/influxdata/telegraf`)/**Prometheus** (`https://github.com/prometheus/prometheus`)/**Grafana** (`https://github.com/grafana/grafana`) (**TPG**) stack (details about how to spin up a development environment are in *Appendix A*). Finally, you will learn how to solve real network operations challenges using the flexible observability stack presented.

In this first chapter, we will cover the following topics:

- Defining network observability
- Describing network monitoring evolution
- Exposing the key aspects of network observability

Defining network observability

Let's go straight to the point: what is network observability about?

To answer this, it's convenient to understand first what network monitoring is because network observability supersedes it. Network monitoring is part of the wide IT operations monitoring focused on the network infrastructure.

Even though you are likely used to the network monitoring term, there is no academic definition of it, and everyone understands it slightly differently. We define network monitoring as *measuring the performance and availability of the network infrastructure*.

Related to this goal, you may be familiar with some of the technologies that have provided information about the operational state of the network:

- **Simple Network Management Protocol** (**SNMP**) polls and traps
- **Internet Control Message Protocol** (**ICMP**) requests (e.g., ping)
- Flow analysis (e.g., NetFlow)
- Packet capture (e.g., tcpdump)
- Logs (e.g., Syslog)

These technologies make up network monitoring, which provides support for diagnostics and service monitoring, with state visualization and alert generation. Network operation teams leverage network monitoring to detect when something is wrong in the network, but this is not enough anymore.

Nowadays, IT operations have raised the bar, and the focus is not only on the infrastructure status but on translating it to the business level. Therefore, observability is about the *end user's experience*, and this encompasses many layers, from infrastructure to applications.

This convergence of responsibilities materialized in the DevOps culture (i.e., bringing together Development and Operations) that coordinates all the IT efforts around the same business outcome. One basic practice is to consolidate different monitoring systems to enable data correlation. The DevOps movement has broken long-time silos in IT departments, and this new collaboration has produced a lot of innovations, which we will explore in this book.

Moreover, it has transformed the reactive approach of traditional monitoring into a proactive one that helps answer handling issues before impacting the services. Ironically, this leads to simpler (but more effective) systems, capable of getting the data to provide the insights that help solve these issues. This is what IT observability is about, helping to identify the unknown unknowns and having a holistic view.

Within this observability realm, network observability encompasses all the technological trends that support the overall IT observability in the network realm.

In networking, this trend toward adopting network observability has been translated to more flexibility in different aspects:

- Interoperable specialized solutions (e.g., open source solutions provide more flexibility)
- More efficient data retrieval methods (e.g., network streaming telemetry)
- More scalable and advanced data processing (e.g., artificial intelligence)
- Richer context and analysis via data integrations (e.g., source of truth integration)

> **Note**
> That being said, we will use both terms (i.e., monitoring and observability) interchangeably in this book, with the same meaning.

This is what this book is about. We want you to understand how to evolve from traditional network monitoring systems to the new network observability approach, tightly connected with the DevOps culture, and how it connects with the other big revolution in network operations: network automation.

Network monitoring evolution

As already mentioned, modern network observability has evolved from network monitoring, a practice that has been in place for several decades. Before delving into the new approach it introduces, it's important to review what has been effective so far and to understand the trends and requirements that have driven its transformation.

What has worked so far

Networks have been monitored to understand their status since the beginning. **ARPANET** (which stands for **Advanced Research Projects Agency Network**), the first packet-switched network started in 1966, had the **Interface Message Processor** (**IMP**) protocol, which provided a few monitoring features. Fast-forwarding some years to the rise of TCP/IP networks, in 1988, the **SNMP** was defined by the IETF (its last version is SNMPv3) to address this need.

SNMP provides a mechanism to manage networks, but it has been mostly used to *monitor* networks, and not to *manage* configuration changes (which have been mostly done via CLIs, until the rise of newer management interfaces). The main characteristics of SNMP can be summarized in a few aspects:

- The UDP transport protocol is stateless, which is useful for state and status polling
- **Management information bases** (**MIBs**) provide structured data to access specific content
- Massive adoption in all network devices, supporting standard and proprietary MIBs

However, not all that glitters is gold, and SNMP has some limitations such as the performance to retrieve large amounts of data and limited coverage for push mechanisms (i.e., SNMP traps).

> **Note**
> This book doesn't cover SNMP in detail (there are many books dedicated to the topic). We will reference it as one of the available methods to retrieve operational data within a holistic network observability strategy in *Chapter 3*.

Similarly to SNMP, event logs using **Syslog** have been widely used, not only for network monitoring but also for applications. Logs are generated when a specific event is seen by the device, and it brings together several pieces of information such as the generation time, the source, the level, and some meaningful message related to the event. This grouping of data is what we refer to as **multidomain data**. This contrasts with the simple SNMP metrics (integers or strings).

And also, pretty common in network analysis are the flow exporters mechanisms such as **NetFlow**, **sFlow**, and **IPFIX**. With some small differences between them, they represent the basic information to define what a packet flow is about, including the source and destination IP addresses and ports, and some other information. Again, like logs, this is multidomain data.

An important benefit of all these methods is their ubiquitous adoption. It's more than likely that any network device you have supports them. However, the implementation of the monitoring solutions, usually in the form of monolithic platforms, makes it harder to combine and relate this data that may be related.

Also, the initial technologies to manage and persist this data, such as **RRDtool**, came with limitations that modern options such as **Time Series Databases** (**TSDBs**) have overcome. Understanding what a modern network observability stack looks like and how to design and build one is the main goal of *Part 2* of this book.

These methods, together with others such as packet capturing or synthetic monitoring (e.g., ping), have been, and still are, solid pillars to build upon a network monitoring strategy. However, the expectations for observability have disrupted the status quo and, in many environments, traditional network monitoring is no longer enough.

Trends and requirements

Here, we summarize the main trends that have influenced and motivated the evolution of network observability:

- Networks are heterogeneous and abstract. Today, networking takes many forms: campus networks, hyper-scale data centers, cloud-based network services, or service mesh. This variety implies supporting different protocols and interfaces and being able to correlate many data types.
- Network operations, following the DevOps approach, have adopted automation to transform how networks are managed. For most network automation tasks, operational data insights are key and require common data models between developers and operation engineers to get a mutual understanding.
- Focus on application performance has become more predominant, and network monitoring needs to contribute to the common view with all the related information. Moreover, microservice architectures increase the complexity of correlating the data.
- Better visibility requires more data, so we need more efficient retrieval methods and data reusability.
- The volume of the data aggregate can be huge, making it impossible to analyze without the aid of artificial intelligence for IT operations (AIOps).

In this book, we will explain the basic concepts to architect solutions to address these challenges (in *Part 2*), and practical examples to implement them (in *Parts 2* and *3*).

Network observability pillars

With all these expectations, there are four pillars that sustain the network observability solutions (depicted in *Figure 1.1*):

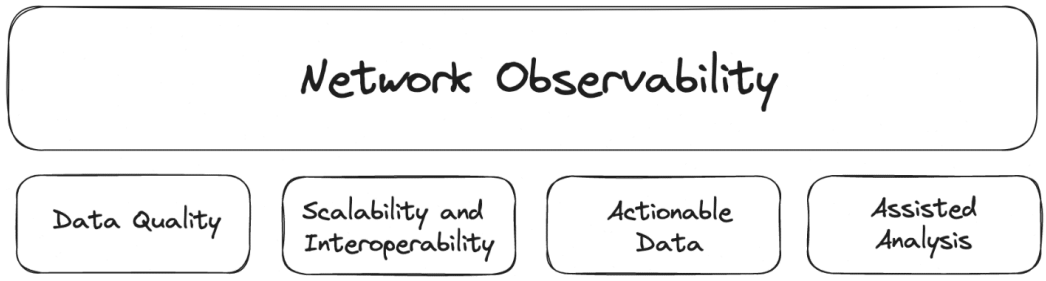

Figure 1.1 – Network observability pillars

We summarize these as follows:

- **Data quality**: Any observation is going to be as good as the data it is based on. We must ensure that the data we collect adheres to some principles that grant its quality.
- **Scalability and interoperability**: To address complex questions, the architecture needs to incorporate specialized tooling that works together in a distributed manner that can scale out as needed.
- **Actionable data**: The active role of network observability within a network automation strategy requires providing data that other components can leverage to act on it.
- **Assisted analysis**: To improve the insights generated, the data needs to be analyzed by machines that can process a large amount of data and applying *intelligence* at scale.

Data quality

It may seem obvious but in any process (even more if it's automated), the quality of the output will be directly proportional to the quality of the input. In network monitoring, operational data is king. Everything depends on the collected data, so it's necessary to carefully select and manage the data that will be used to generate the insights.

But what is data quality exactly? There are many definitions of data quality, but all of them pivot around a few dimensions:

- **Relevance**: Is the data useful (and used) for the purpose it was collected?
- **Accuracy**: Is the data precise enough to give insights into it?
- **Timeliness**: Is the data current enough to provide almost real-time conclusions?
- **Comparability**: Can the data be compared with other datasets?
- **Completeness**: Are there any records missing?

In the context of network observability, we could summarize **quality data** as *the data that is fit for answering the relevant questions about your network*.

This book doesn't cover the data quality topic in depth (there is a myriad of books only focused on this topic), but through this book, we will implicitly refer to data quality characteristics through many sections of the book, as in these instances:

- In *Chapter 3*, we will introduce the relevance of **streaming telemetry** to get almost real-time metrics from network devices. We will be tackling the *timeliness* dimension.
- In *Chapter 5*, we will explain the importance of **normalizing data** from different sources to make it *comparable*.

- In *Chapter 6*, the concept of **data enrichment** increases the data's relevance, adding more context so data can be easily correlated (not only by its timestamp).
- In *Chapter 7*, when explaining the **persistency layer**, we will tackle how some databases can fill missing data records with probable data to help run processing on top, helping with *completeness*.

Scalability and interoperability

Traditional network monitoring systems have been implemented as monolithic ones performing all the necessary functionalities (e.g., collection, storage, and visualization). This all-inclusive approach may seem convenient to get started. However, as more specialized features are required (e.g., a new database type or a new collector agent), it becomes evident that it's unlikely that one tool would be capable of addressing all your needs.

On the contrary, we need to acknowledge that evolving the network observability stack requires plugging in different components, specialized on some of the functionalities. These distributed architectures require clear interfaces to connect the different components, and a solid orchestration to deploy them in a repeatable manner at scale.

Moreover, to make this composable approach efficient, we should avoid duplicating roles. For instance, we have seen many cases where, to leverage the analysis from two different monitoring tools, each one has to run SNMP collectors and collect the very same data. Why not *collect it once, and reuse it many times*?

Related to architecture requirements, the new stack has to allow high scalability to handle the increase in the amount of data and processing required to provide the expected insights. This scalability should allow per-component upgrades (i.e., scale out) instead of the whole stack required by monolithic systems (i.e., scale up).

We will cover these topics in *Part 2*, from a general introduction of the recommended stack in *Chapter 4* to per-component details in the other chapters.

Actionable data

The adoption of network automation has revolutionized how networks are managed. This movement started with the software-defined networking hype (do you remember OpenFlow?) around 2010, and, from there, it evolved in different ways, strongly influenced by DevOps practices (some people refer to it as **NetDevOps**). When we talk about network automation, we refer to replacing the manual changes on the network infrastructure (e.g., the CLI in network devices or the GUI on controller-based ones) by using a repeatable approach that could replace (most) human intervention.

Network automation is a big topic that we don't pretend to cover in this book. However, in every chapter, we will highlight the role of network observability in the network automation space.

One key requirement for network observability within automated network operations is the need to provide actionable data to feed into closed-loop systems (i.e., systems that automatically adjust depending on their output). As we will see later, network automation's heart is about defining the *intended* state of the network that drives the whole system, from defining the configuration and the operational state. Using this as a reference, the network observability will collect the *actual* state and check whether some kind of mitigation action is needed or not.

Part 2 of the book covers how to design and implement the functionalities that support automated operations, and in *Part 3*, we explain how to leverage the recommended stack to solve real problems.

Also, you should not forget that the network automation solutions need to be observed themselves to understand how they behave and influence the network state.

Assisted analysis

Network monitoring has always supported humans with data to understand patterns or provide insights about how the network services were running. Since the early stages, network monitoring has also aspired to provide capable insights on top of the data collected. However, until recently, the analysis of monitoring data had limited (but still useful) use cases. For instance, you have likely defined threshold alerts in your monitoring systems to raise an alarm when the CPU level exceeds some threshold.

Nowadays, with the massive scale of IT systems and their complexity, the need for assisted analysis is more relevant. Answering this call, the advent of **AI/ML** (**Artificial Intelligence and Machine Learning**) technologies has transformed the game, and the promise of implementing **Artificial Intelligence for IT Operations** (**AIOps**) is becoming a reality.

AI/ML provides solutions to various problems, including the identification of data clusters with shared characteristics, such as the number of **access control list** (**ACL**) hits according to device role, and the detection of anomalies by comparing current metrics against historical data while considering seasonality. Even more popular, **large language models** (**LLMs**), used by chatbots such as OpenAI's **ChatGPT** (`https://openai.com/chatgpt`), allow reproducing human language processing to provide complex answers based on all the previous training. For instance, you can ask for the potential impact of a log message that has not been classified before in your system to get an educated insight into the related implications.

The potential of these new tools to contribute to the analysis of network operations is immense, but so are the challenges. There is a learning curve to understand when to use one or another technique, and more importantly, how to select and manage the data (including anonymizing it before sharing it with an external system) to achieve the desired results. We won't go deep into the foundation of ML/AI, but in *Chapter 13*, we will provide some inspirational examples of leveraging them to improve the insights produced by network observability.

Benefits

After this brief introduction to network observability, we want to finish it by highlighting some of the key benefits it brings to the table:

- **Reduced time to solve incidents**: More and better data is available for deeper analysis of multi-dimensional issues that affect the services running on top of the network infrastructure, providing educated suggestions to resolve the issues.
- **Better end user experience**: Due to the shorter time to identify users' issues, and also including the user's perspective in the analysis.
- **More accurate capacity planning**: By combining all the data generated, it is possible to reduce the over-provisioning of network services tailoring to the actual needs.
- **Accelerated network operations**: Being able to validate the state of the network against its intended state enables faster configuration deployments that are validated by observability. It also supports canary deployments, where a big network change is only rolled out to a small subset of the network to reduce the blast radius effect, and incrementally rolled out to the rest once the state is validated.

Understanding the key network observability pillars and the benefits they provide will help you navigate through this book. When we present our proposed architecture and stack in *Part 2*, you will notice the influence of these pillars behind every recommendation.

Summary

In this chapter, we presented the network observability topic and how it evolved from traditional network monitoring. We looked at its evolution and the main trends that have influenced its transformation. Finally, we introduced the four pillars on top of which we will build the observability stack, and some of the expected benefits of this approach.

In the next chapter, we expand on the importance of providing business-level insights and the role of observability in modern network operations.

2

Role of Monitoring and Observability in Network Infrastructure

As previously introduced in *Chapter 1*, the relevance and scope of observability in IT infrastructure goes beyond networking, and most of the ideas and topics proposed in this book are easily portable to other infrastructure realms. However, we want to provide a closer look at networking and its own needs.

For this reason, before going into the pure observability topics, we think it's important to give you a high-level overview of how networks have evolved in the last decade. With this context, it will be easier to later understand the requirements that we expect from a modern network observability solution.

Also in this chapter, we present a shift in how you understand your network. We encourage you to see it as a *product* – either it has a direct impact on your company's revenue or it's supporting it. This mindset will guide how we approach the transformation of the raw data we get from the network, in the form of metrics, logs, and other types, into actual information that can help others set expectations about how to consume the network services.

The chapter covers the following topics:

- The state of networking in the 2020s
- How to transform data into information
- Recapping the expectations for network observability

Let's start doing a quick recap of what networks look like nowadays.

Networking in the 2020s

We have to admit that networking is not the spearhead of rapid innovation in IT. There are good reasons for that. Networks require proven interoperability that has been usually sustained in standards that may take a long time to implement, and the blast radius of a network failure is worse than an electric power outage (no *battery mode* exists). Thus, networking has been resistant to changes until it has become totally necessary to support the evolution of applications running on top.

Changes in networking in the last 20 years have taken many flavors, both technological and cultural, and it has impacted the expectations in network observability. Even though this book is not about network architectures or solutions, we believe that having a 10,000-foot view of these factors will help you better understand the impact of the observability solutions around.

Technological changes

Everything depends on the perspective. If we use the 1990s as our reference, we could say that the current networks forming the internet are more homogeneous than before because the network protocols converged into a few of them, such as IP, TCP/UDP, or HTTP. However, it's also true that today's networks are no longer built only around closed boxes by a few vendors and are adopting open Linux **operating systems** (**OS**) and virtual network services running on different platforms, making these networks more heterogeneous.

These networks have evolved in different directions, depending on which was the main purpose. We can find network service providers that connect many networks, campus networks that provide access to end users, or data center networks that support backend applications connectivity. Moreover, different nuances are depending on the application nature they are supporting, such as the fintech use case, where a stable and low latency is crucial, to video content delivery, where multicast support allows scalability.

Without going into low-level details, we are going to analyze a few of the most relevant technological changes, starting with how the architecture of these networks has changed.

Network architectures

The authors of this book have been managing networks since the 2000s. We can still remember the days when the networks were connecting a *bunch* of clients (i.e., PCs/personal computers) to a *few* servers (in the order of hundreds at most), and finally providing internet access.

In that context, a popular architecture stood out, the **three-tier architecture**. The name comes from the three layers that compose it: the *core*, *distribution*, and *access* layers. It has been, and still is, the most common architecture (with different variances) for multi-purpose **Local Area Networks** (**LANs**)

because it works well for the north-south traffic pattern (i.e., when most of the traffic goes from one access layer into another domain up in the architecture):

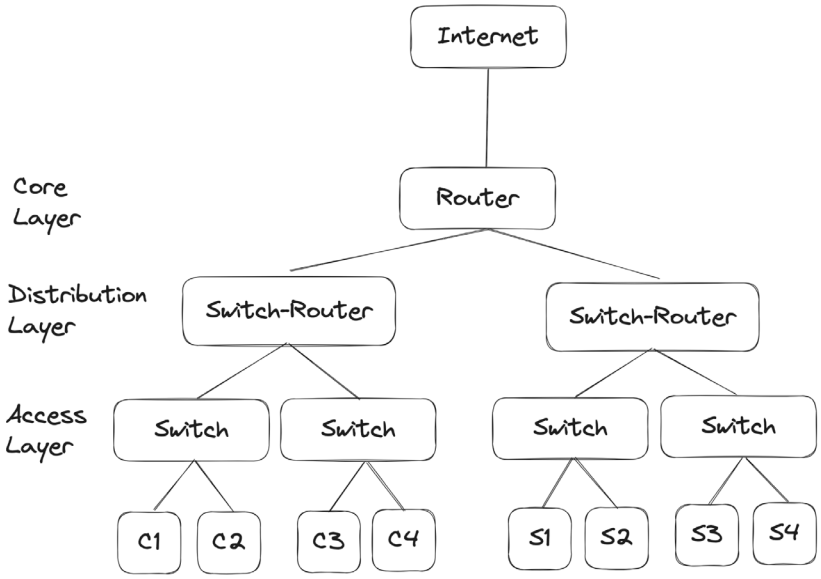

Figure 2.1 – Three-tier architecture

In large scale companies, such as Google or Meta, it became evident that this architecture had scalability limitations to support new application traffic patterns. The new network architecture had to provide a predictable network latency, a higher connectivity capacity, and an easier way to scale it via scaling out (i.e., adding new devices to the network) instead of scaling up (i.e., replacing devices with ones with more capacity). This architecture is known as **leaf and spine** (though, in some cases, it may have different names), but the actual concept comes from the old Clos network design, which provides a consistent flattened network design.

Many articles and books have been written about these topics, but a blog from Meta (formerly Facebook) in 2014 (`https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/`), about the data center fabric network topology, was one of the first explaining it to the public. In the blog, you can notice how the design follows a regular pattern replicated at different levels, and there is always the same distance from one server to another.

On top of this standard architecture, different *virtual* networks can be constructed. The previous is just the underlay network, but on top, many different overlays can be built to provide custom networks. This separation of concerns (i.e., underlay and overlay) allows more flexibility in creating networks dynamically, but it also increments the dimensions to consider about the network state, as there is not only one network but a combination of many.

> **Note**
>
> A common implementation of this architecture in the data center is based on **Ethernet VPN** (**EVPN**) and **Virtual Extensible LAN** (**VXLAN**) protocols (i.e., EVPN controls how the overlay VXLAN tunnels are created). In the **Wide Area Network** (**WAN**) area, the **Software-Defined WAN** (**SD-WAN**) solutions use a *controller* to orchestrate overlay connections between branches connected over the internet (the underlay).

Both network designs are still in use because they serve different purposes, and in many environments, the network is a combination of both. One way or another, what is a reality is the higher number of connected devices (think about IoT devices, for example), thus, a higher number of network devices (i.e., switches and access points) to connect them (which need to be monitored).

Virtualization in networking

Twenty years ago, servers and network devices were directly mapped to physical boxes. Every box was running proprietary applications running on top of closed OS, consuming the hardware resources directly.

In the late 2000s, the virtualization of servers became mainstream leading to a much flexible way to provision new servers on the same physical box. Running different OSs on top of a hypervisor (abstracting the physical resource) allows OSs to think that they run alone while they are sharing the same physical host.

This transformation on the server side motivated the appearance of similar solutions for networking, and other trends, such as **Network Function Virtualization** (**NFV**), which pushed toward taking network functions traditionally implemented in hardware into software.

In addition, containerized solutions came into play as an interesting option to create development environments, which also require observing the resulting state before changes are accepted.

> **Note**
>
> In this book, we use containerized network OSs to create the lab environment. More details are in the *Appendix A*.

You can already notice how all this flexibility requires dynamic network monitoring that is not statically set once and kept for the whole life of the device. It has to change dynamically as the network changes.

Network automation

If you have been networking for a while, likely, you are mostly managing your network via the famous **Command-Line Interfaces** (**CLIs**), which provide a human-readable language to define how the network should behave. This worked well for small-scale networks, but as we already mentioned, with the increase in size, and heterogeneity, of modern networks, manually operating network devices doesn't scale and has other limitations.

In this context, around 2010, we witnessed two new paradigms that eventually converged into what we understand as network automation:

- Rise of the **Software-Defined Networking** (**SDN**) movement. It proposed that the network behavior should be managed via software
- DevOps culture, where development and operations are deeply connected, so the software and infrastructure evolve closer

In the present day, most of the new networks are managed in an automated way. Instead of using human language constructs via CLI commands, the network devices have adopted new interfaces for software management **Application Programming Interfaces** (**APIs**), such as NETCONF, RESTCONF, **gRPC Network Management Interface** (**gNMI**), or common-purpose REST APIs. These interfaces also came with new features to support advanced observability use cases. The data (configuration and operational) come modeled, usually using **Yet Another Next Generation** (**YANG**), which enables more effective data processing and also provides model-driven telemetry (i.e., a continuous flow of data) to get data at a higher rate (more details in *Chapter 3*).

Network observability has a key role in this context because it's no longer a passive component that only monitors the network's operational state. It is taking a step further and being the catalyst of the automation tasks to mitigate the network when issues (i.e., divergence between reality and expectations) are detected.

> **Note**
>
> The *Network Programmability and Automation, 2nd Edition* book by authors Matt Oswalt, Christian Adell, Scott S. Lowe, and Jason Edelman provides a high-level architecture of the role of network monitoring in a complete network automation solution.

Linux networking

Traditional network boxes have been running proprietary OS with null or very limited access to all their capacities. This is the reason most of the network monitoring has been done off the box instead of running a software agent within the box to collect operational data, such as has been done with servers.

In the early 2010s, new incumbent players transformed this rule (e.g., Cumulus Networks and Arista). They started to run network functions on top of a Linux OS while allowing access to running processes directly in the box – in some cases (e.g., Cumulus), without any strong coupling with the hardware platform.

This openness has allowed running the network OS on many different platforms, and the appearance of different network OSs disconnected from specific hardware vendor such as SONiC or VyOS.

In terms of observability, this means that we are no longer only interested in the network control state but also in the state of the OS, and the process running in the box.

Cloud networking

Around the same time, the IT industry was radically transformed by cloud services where the IT infrastructure services (e.g., compute, storage, and networking) were abstracted and provisioned on-demand.

This shift in the way of interacting with IT infrastructure allowed a more rapid go-to-market of new applications as you no longer need to provision and manage your infrastructure directly. Instead, you can leverage APIs to embrace the **Infrastructure as a Service** paradigm and start paying as you grow without worrying about capacity management.

The adoption of cloud services (private and public) is massive today. And, as you may infer, it also involves cloud networking services that need to interact seamlessly with the physical network infrastructure. Most of the network environments are hybrid. This heterogeneity increases the complexity of what to observe and correlate to understand the actual state.

On top of this, if your company is actually *running* the cloud, you must be ready to manage the underlay and overlay networks to support it. For example, if your company manages its own Kubernetes cluster, the network observability has to cover it to provide a complete network state coverage.

All these technological challenges have transformed the network with all the new requirements. However, these changes came along with another important transformation in terms of cultural changes.

Cultural changes

The biggest cultural shift in IT infrastructure has been the change from a slow and static provisioning process to a fast-changing and dynamic approach. Everything in IT is consumed *as a service* (via APIs). We have many flavors, but we can simplify in two:

- **Software as a Service** (**SaaS**), where you simply use an application without caring about how it is built or operated
- **Infrastructure as a Service** (**IaaS**), where the computing, storage, and networking resources are consumed without having to buy, transport, rack, and connect

> **Note**
> IaaS is not magic. There are teams behind these services who do the *real* work to allow others (the users of the IaaS) to consume it with this paradigm.

Regarding networking, the one that applies to networking is the IaaS, which we could name as **Networking as a Service**. Aside from the technical challenges (which we introduced earlier), the key points of the cultural transformation are as follows:

- Network users want to get their requests fulfilled in the order of seconds or minutes, not days or weeks as we used to.

- No one (i.e., users) cares about the network heterogeneity (i.e., physical and cloud network services). People just want the network to work, and this requires offering a proper abstraction.
- Adding human interventions slows down the process and only automated networks can implement this paradigm properly. Adopting automation and all its implications is no longer an option.

All these drivers led to establishing the network as a product (or a bunch of them, depending on the different purposes they have). And, when something becomes a product, it has to be managed as such. This is why we move next into how we evolve from just gathering operational state data into transforming it into business-level information that represents the state of the network as a product.

Transforming data into information

A key aspect of the modern network observability approach is to focus not only on collecting and visualizing operational data but also, on going a step further, transforming the data into actual information with a purpose.

The importance of using business terms

Networks (and most IT infrastructure components) have been seen, in many cases, as necessary IT actors without a strategic role in organizations' businesses. Despite being a crucial actor in sustaining almost every part of every organization, IT infrastructure departments haven't been able to properly communicate the value they provide

IT infrastructure teams have to change this to become more relevant in the business strategy. The business' success is built around all the teams, but some of them can explain more clearly how they contribute because they speak the business language.

So, the question is, how could we speak the business language? We recommend focusing on translating the network information into something that can be translated into business terms. For instance, the next table shows a few examples:

Business Goal	Network Operational Data
How much impact a network incident has on the revenue	How much network downtime there was and which business services were impacted
How much better users' experiences are, and how this translates to customer satisfaction	The level of packet loss and latency seen by the end users
Control the revenue per utilization of services	Use network stats to charge users per consumed bandwidth
How much faster the company processes can be delivered and contribute to increasing the company revenue	How many times a network service is automated versus run manually

Business Goal	Network Operational Data
How much spare network capacity is available to expand the business?	How many access ports are available, and how much capacity in the uplink links is left
When will we need to increase the capacity of the network according to the current usage trend?	Use data forecasting analysis to infer the trends of data consumption and determine the breaking point in the future.

Table 2.1 – Business goals mapped to network operation data examples

Mapping the business goals into network observability data is a key task that shouldn't be procrastinated. In this section, we will use **Key Performance Indicators** (**KPIs**) to help others understand how good (or bad) the network services are doing, and how these impact directly to the business.

Defining KPIs

KPIs help with translating data into information. In the networking context, KPIs transform network operational data into something that can be understood by other teams, technical or not.

There is a lot of literature about KPIs as they may apply to everything. In the service level terms (remember that we want people to see the network as a product/service), it's useful to introduce three basic service-related concepts:

- **Service-Level Indicator** (**SLI**): The metric that defines the quality of the service perceived by the users.
- **Service-Level Objective** (**SLO**): The target goal for our SLI.
- **Service-Level Agreement** (**SLA**): The SLO translated into a formal commitment. If it's breached, legal claims are supported.

The following figure shows the order of how the levels of service are defined:

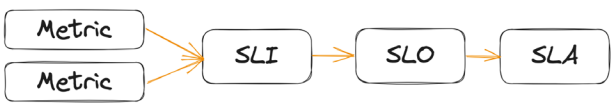

Figure 2.2 – How to define the levels of service

Note

You can get more information about service level definitions in the Google SRE guide: `https://sre.google/sre-book/service-level-objectives/`.

Let's get now into each one's details to map them into network examples, starting from the beginning.

SLI

You can't commit to something (e.g., an SLA) without understanding what you can measure. In networking, an SLI measures some aspect of your network state in terms of the service it is providing. For example, a common SLI is the availability of your network service.

However, the *availability* SLI is an abstract term that requires different data depending on each use case. A network may be available in some terms, but every network has different expectations for the service it provides. For instance, knowing that the end-to-end reachability is working may not be enough. You may want to ensure that the communications are below a certain level of packet loss or average latency. Therefore, the *availability* SLI may be composed of a few metrics that combined create the final indicator:

- Average percentage of packets lost on a transfer
- Time spent by packets moving from one end to the other in the 95th percentile

> **Note**
> All these metrics must be related to the defined interesting traffic. This means that the packets may be related to the traditional ICMP protocol, or a synthetic application running on a specific network port (e.g., `TCP/443`) with some traffic pattern.

The list of metrics and other operational data could go on, adding more and more dimensionalities to define the SLI. This is a trade-off; more data would require more resources, meaning more complexity to process, and sometimes not adding relevant information. Thus, not overcomplicating but getting the right level of data is key to successfully defining the SLIs of your network.

Once you understand the power of SLIs, you may try to transform every metric into an SLI. This will overwhelm you. We recommend focusing on relevant information that adds value, and not over-monitor. The question, then, is "*How do you know what adds value?*" Well, put yourself in your users' shoes, and try to understand the key indicators that will help them understand how your service is expected to behave, so they can make decisions on top of them.

When you know which information to track (i.e., the SLIs), it's time to define what is the expected value of the SLI that you want to target, the SLO.

SLO

The SLO is the target value of your SLI – in other words, the value of the SLI that you want your network service to match. The SLO is not a hard commitment but a soft one that should communicate, internally and externally, what the network service is aiming for. Externally, it helps your users understand what they can expect from your service, and internally, it helps your team work toward building a network that delivers the expected level of service.

Continuing with the previous example about the network availability, once you have set the SLI measurement in place, you will start having a history of data to understand the actual value of your SLI over a while. For example, in the following graph, you can observe 12 data points of the availability SLI ranging from 94.5 to 100 over a period of time:

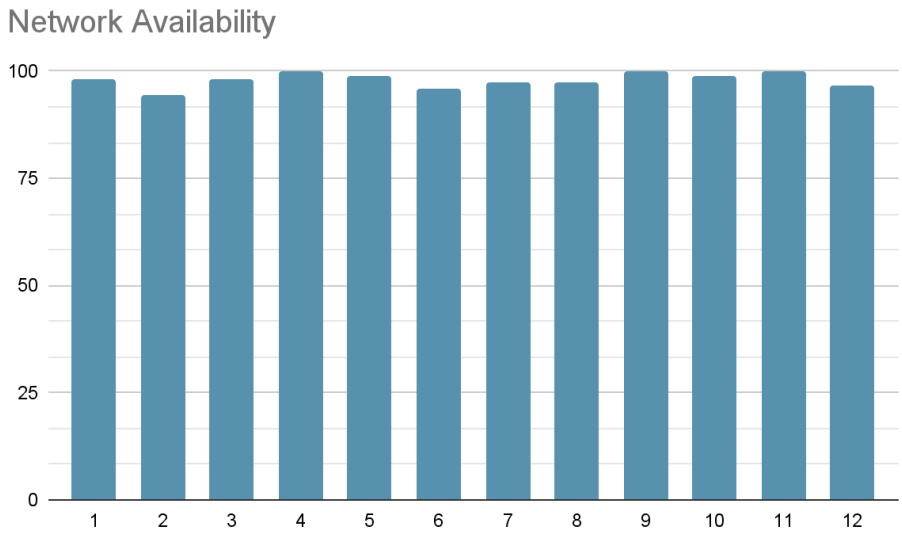

Figure 2.3 – Example of network availability graph

Having a reference is crucial to understanding the actual level of service that you may offer. You should target an SLO that is realistic (or achievable) to align expectations from the internal team and the external users.

But, what is the value you should set for the SLO? In the most conservative approach, you may look for the worst SLI value, but this could be impacted by some bad times. For example, in the previous example, the lowest number is 94.5%, and it could be a valid SLO to communicate externally. However, internally, you could work toward a higher one as the end goal is to improve your network service. In the graph, we could target 97.5% as the internal SLO that is achieved in 70% of the samples, and work toward it.

Tip
Overdelivering is always better than overcommitting.

Every SLO comes with a cost in terms of resources and attention. It's important to choose carefully the right level of service for your network, neither more nor less. This means that having an SLO of 99.99999% may come with a cost that is not necessary for your users because a 99.5% availability is enough to run their business applications (which are resilient to transient network issues).

The SLOs are useful to explain the network roadmap. Every decision you take on the network design and operation will have an impact on them (positive or negative), so relating one to the other can help you justify your proposals.

Finally, when your network service is used by others (internally or externally) and you want to commit to them (usually, external users), you have to evolve from the SLOs into the SLAs.

SLA

The SLA goes one step beyond the SLO. It's no longer a goal, it's a commitment, which, if not matched, may originate legal actions against your team. So, you should take it seriously, and require support from the legal department to write SLAs.

However, you won't always need to provide SLAs. Only when your network users require it. It's pretty common within companies to use only SLOs as a soft commitment without official SLAs.

SLAs are useful to define the right level of expectations for the customers, so they can use the network service with enough confidence. This usually takes a pyramid approach. To define the SLA for your network, you may require other services that will come with their SLAs (e.g., external network services such as WAN lines, or virtual network services in the cloud), and your final level of service is influenced by these SLAs and the way you design your network. Your work as a network engineer is to build/architect network services that match your SLAs, taking into account the others.

If you have been reading through, it shouldn't come as a surprise that the SLAs should be less ambitious than your SLOs (if you don't want to get buried under many legal issues). Why? You shouldn't commit to something that is beyond your real capabilities.

SLAs are legal contracts so they have to be concise and clear to understand by others. Also, even though SLAs are usually based on aspects that you control, they are influenced by other external elements. For example, there may be some issues that require consumer involvement to resolve them, and this must be clearly defined.

The SLAs transform your network into a product upon which others can rely. Thus, in most cases, if you want your network to be taken seriously, you have to get to them (or robust SLOs if it's an internal service). However, you should wisely define them to provide the expected service level without putting too much pressure on the team. Ideas such as an error budget can help with that.

Error budgets

You always want to provide the highest SLA to convince your network users that you are providing a great service. But adjusting the SLA or SLO too close to your actual best SLI is putting a lot of pressure on your team because you have little margin when something goes wrong.

The Google SRE guidelines (referenced in a note earlier) recommend creating an **error budget**, or in other words, adding padding between what you commit and what you can achieve. This gives you some extra downtime margin to help you identify unknown situations and improve, while you still match your users' expectations.

In addition, don't overachieve! It's common in infrastructure services such as networks to provide a consistently higher level of service than the actual SLO. However, when this situation extends for too long, your users will get used to it, and they could assume that your performance is higher than your actual SLOs, creating an over-dependency on it. It's not a bad idea to use your error budget in this case, and introduce some turbulences (without breaking your SLO) to *encourage* your users to build a more resilient service around.

A related approach is the **Chaos Monkey** idea, promoted by Netflix, which created some jaw-dropping effects in SRE conferences in the middle of the 2010s. The idea was simple and radical: instead of waiting for the relying infrastructure to fail (in their case, AWS cloud services), let's introduce random issues so the application running on top must adapt from the beginning to unexpected service level conditions.

From data to information

So far, we understand that defining our service levels in business terms helps everyone understand the value of the network as a product.

However, the operational data extracted from a network needs a curation process before turning it into information that helps others understand how the network is running.

This book is all about this. We will focus in *Chapter 3* on understanding which types of operational data are available, and in *Part 2 and Part 3* of the book, how to collect, process, store, visualize, and act on this data to properly communicate relevant information to different audiences.

The audience of a network observability solution may vary from internal network engineers to other IT departments, or even business-related teams. Everyone will have different expectations, as we will explain in the next section.

In all cases, the process of transforming data into information is putting ourselves on the receiver's side, enunciating the questions the receiver may have, and then coming up with the proper answer to that question.

> **Note**
>
> An interesting approach to consider is the **Goal Question Metric** (**GQM**) approach about reusing the questions and metrics for different business goals. You can read more here: `https://www.cs.umd.edu/users/mvz/handouts/gqm.pdf`.

For example, in a content delivery internet network, providing a trend of network transit link utilization correlated with the utilization of compute ports could lead to different business decisions, such as increasing the transit link capacity (if there is the availability of access ports) or to look for a new location to create a new **Point of Presence** (**PoP**) if we are reaching the capacity limit of compute ports of the network architecture per site.

Having introduced the context of networking in the 2020s and how we can transform the data into relevant information, the final section of this chapter will present you with the most relevant expectations for network observability in this context.

Expectations for network observability

The focus on useful information, combined with the key role of network observability within the automatization of network operations, has raised the expectations that these solutions should fulfill.

Heterogeneous and enriched data

As we have mentioned in the *Technological changes* subsection, the networks are more diverse nowadays. This means that the data obtained from the different network systems and platforms is no longer as it was in the past. We have been used to leverage mostly on **Simple Network Management Protocol** (**SNMP**), and its standard **Management Information Bases** (**MIBs**), which made the data gathering and processing pretty consistent.

Today, the information we are looking for comes from the combination of operational data of different types. For example, you may have a **Border Gateway Protocol** (**BGP**) session running between an on-premises router connected via VPN to a cloud provider. On each side, you have to use the proper collection methods (e.g., SNMP, gNMI, CLI, REST API, logs, etc.) to get the operational data, which is not usually directly comparable.

Moreover, it not only has different types of data to be compared but also to be combined to increase the understanding of the network state. For example, combining the state of the BGP peering with the logs from the interface under which the peering is established can provide more context to understand what's going wrong.

In modern network observability, the flexibility to integrate many different types of operational data, and to enrich the information by combining them, is a must.

Proactive role in network automation

Traditional network monitoring was mostly focused on providing good visualization and reporting, and notifying alerts when some conditions were met (e.g., threshold violated). That's still in the current scope, but within the scope of network automation, network observability requires a much more active role.

As we will see in *Part 2* of the book, the key role of network observability within a network automation solution is to create the feedback loop between what we expect the network to be (i.e., the intended state) and what it is (i.e., the actual state).

To achieve this, it is necessary to integrate with the network source of truth that contains the network's intended state. The source of truth is an abstract concept that encompasses all kinds of data that defines the state we want the network to be. It can be represented by simple data-structured files (e.g., YAML or JSON) or by sophisticated systems backed up by databases.

Comparing both states, intended and actual, the network observability solution should connect to other systems (e.g., a workflow orchestrator) to start the automated process of moving the network back to the desired state or breaking a continuous deployment process to release new changes if something is wrong. All these flows should ideally be automated, but they could also contain some manual judgment steps when needed.

Full visibility of network state

It's an industry-accepted rule to focus on the **golden signals** to monitor an application or a network. These golden signals are the latency, the amount of traffic, the errors, and the saturation. These signals provide key information to understand the state of your network, and they have been used in network monitoring for years.

> **Note**
>
> In this book, you will see concrete network-related examples of what you should care about when collecting metrics and generating alerts.

However, networks have some characteristics that require an extra level of visibility. Networks are complex distributed systems that run protocols between the nodes, and some of these nodes are out of your control (i.e., BGP peering with another network). Also, the network transports many different types of applications on top, and each one may have different behavior depending on the network configuration and its needs (i.e., using **Quality of Service** (**QoS**) per application, or having firewall rules blocking some traffic.

Thus, to get capable information about the network state, it's necessary to get data from many sources to provide assurance that the network is behaving as expected for all the different use cases it is expected to support (i.e., you can't control 100%, focus on what you are committed to). To achieve

it, network observability would require collecting and combining different types of data, as we will explore in the next chapter.

Faster, more accurate, and at scale

So far, we have already introduced some hints about the explosion of data to be considered, and how this data needs to provide information that eventually will become knowledge. Here are some of the expectations:

- Bigger and more heterogeneous networks, with more nodes and different platforms
- Useful SLIs combining different data to provide a more complete visibility
- Active role in network operations via network automation
- More data samples due to a higher frequency of data retrieval

If we try to delegate to humans the task of processing the data to manage these expectations, you easily understand that it's not doable. For example, in a network with thousands of interfaces, how could you correlate an unexpected traffic pattern in real time and be able to conclude the root cause via understanding the dependencies?

For sure, you could try to create a logic at a small scale that works, but to apply it at a bigger scale would require using **Artificial Intelligence** (**AI**) and delegating this work to computers. This is not new; we have used AI for a long time. The simple **If This Then That** (**IFTTT**) approach is AI. It's a logic that a computer can apply. However, using simple logic for AI has some limits, as we have to provide concrete patterns to be understood by computers. The next iteration is to allow the systems to learn from historical data and define knowledge models that can help infer future patterns, not explicitly defined. This is known as **Machine Learning** (**ML**).

In network observability, we can find many use cases where AI/ML can help to provide educated suggestions (notice that learning is never 100% accurate). For instance:

- Use general-purpose LLMs to obtain log summarization and to propose a potential **Root Cause Analysis** (**RCA**)
- Leverage dynamic traffic thresholds forecasting what the threshold under some conditions would be
- Determine traffic patterns on interfaces to detect anomalies (e.g., higher or lower bandwidth for the upstream/downstream interface types)

These are just a few examples; we will cover more AI/ML use cases in *Chapter 13*.

With this we end this section, in which we have introduced a few of the most relevant new expectations that modern network observability solutions have to face.

Summary

In this chapter, we have briefly introduced the state of networking nowadays to understand how this impacts network observability and then presented the challenge to transform the network operational data into information that the business can understand to realize the role of the network in the company operations. Finally, taking all into account, we have enunciated the top expectations for network observability, which will be addressed with specific examples in this book.

Next, to conclude *Part 1* of the book, we will provide more context about what modern network observability means and the different types of operational data that may be in its scope.

3

Data's Role in Network Observability

As we introduced in *Chapter 1*, network observability aims to provide answers to many different questions by getting visibility of the operational state of the network, from different perspectives. This visibility comes from many distinct types of operational data, each one requiring different collection methods.

In this chapter, you will learn about the following:

- The basic concepts around traditional network monitoring and the new telemetry approach, together with a classification by which network planes (i.e. management, control, and forwarding) are targeted by each monitoring protocol.
- Recap of available data types and how to collect them with programmatic hands-on examples leveraging Python libraries. Having a basic understanding of Python will help you navigate the examples better, but the explanations should make it understandable either way.

> **Note**
> We use the informational RFC 9232, *Network Telemetry Framework* (`https://datatracker.ietf.org/doc/rfc9232/`) as a reference in this chapter because it offers a solid framework and taxonomy of this topic. Please, check it out.

This chapter should serve as the basis for getting into *Part 2* of the book, where you will see how the collected data is being processed, stored, and consumed to realize the goal of network observability. However, due to scope limits, we won't cover all the options described here in the rest of the book.

> **Note**
> In this chapter, we will only focus on the protocols, not on the complete architecture of the observability solutions, which we will tackle in *Chapter 4*.

The chapter covers the following topics:

- Network monitoring and telemetry
- Collecting data, in practice

Let's start by understanding the evolution of traditional network monitoring to network telemetry.

Network monitoring and telemetry

Network monitoring, as you already know, is not something new. Network administrators have been gathering network operational data for years to understand how the networks behave. This approach, which we could define as classic network **Operations, Administration, and Management** (**OAM**) methods, has encountered some challenges that stand in the way of meeting current expectations. To address some of the limitations of the previous approach, a new technical approach, network telemetry, has emerged with different techniques.

It's not one or the other. To support modern network observability, we will leverage all the available options. So, let's start with a recap of the challenges of traditional monitoring.

Challenges of traditional network monitoring

When someone asks you about network monitoring, the first two protocols that will come to your mind are probably **SNMP** (for metrics) and **Syslog** (for logs). Both have been the foundation of every network monitoring system. Then, other OAM methods, such as Ping, Traceroute, and BFD, have complemented them to get complementary insights.

Even though these protocols have been serving monitoring purposes relatively well, there are a few problems that have not been fully solved by them:

- Poll-based low-frequency data collection doesn't provide enough data or precision for some monitoring applications
- Heterogeneous operational data (e.g., user flows, device configuration, and line card metrics) are only covered partially, and traditional network devices are not programmable enough to adapt to
- Some solutions (e.g., Syslog or **command-line interfaces** (**CLIs**)) lack a formal data model, and machines can't work without structured data (well, with ML techniques, they can at least try)
- Data push solutions have limited capabilities with some predefined management plane events (e.g., SNMP Trap) or sampled user packets (e.g., sFlow)

The network industry has been aware of these limitations for years and has worked to solve them with new protocols that we include in the network telemetry umbrella, which we will describe in the next section. However, do not disregard these solutions; they may still be the only way to get data from some network platforms, so there is still a role for them in your observability solutions.